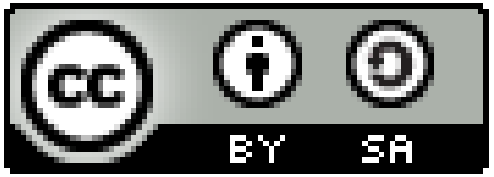
How much is tracing?

Measuring the overhead caused by the tracing infrastructure

Andreas Klinger

IT - Klinger
http://www.it-klinger.de
ak@it-klinger.de

ELCE 2023, Prag
30.06.2023



The author is to be mentioned as:
`Andreas Klinger <ak@it-klinger.de>`

1 Tracing overhead

- Scope of measuring
- Measuring with kernel driver
- Kernel configuration
- Ftrace infrastructure
- Hist trigger and synthetic events
- Histograms with eBPF
- Kernel probes
- Function and function graph plugins
- trace printk
- Summary of kernel tracing overhead
- Userspace tracing
- Measuring gpio latencies

Can we trust the measured time values?

```
             |  irq_enter() {
             |    tick_irq_enter() {
   2.583 us  |      ktime_get();
   1.750 us  |      nr_iowait_cpu();
             |      tick_do_update_jiffies64.part.0() {
   1.458 us  |        calc_global_load();
             |        update_wall_time() {
             |          timekeeping_advance() {
             |            timekeeping_update() {
             |              update_vsyscall() {
   1.833 us  |                flush_dcache_page();
   5.917 us  |              }
   1.709 us  |              raw_notifier_call_chain();
 + 14.125 us |            }
 + 20.500 us |          }
 + 24.208 us |        }
 + 32.166 us |      }
 + 45.042 us |    }
 + 52.041 us |  }
```

Motivation

- Performance optimization
- Estimation about tracing overhead compared to a production kernel without any tracing
  $\rightarrow$ Can we afford to deliver with tracefs compiled in?
- Kernel configuration:
  With preemption (`(PREEMPT)`) or
  with realtime (`(PREEMPT_RT)`)
- Usage of the facility through
  the tracefs (directly, trace-cmd, perf) or
  via eBPF also makes a difference

All source code, configurations, oscilloscope dumps and presentation available:
url: nc.it-klinger.de
usr: elce
pwd: elce062023

Tracing facilities for kernel drivers

- Tracing event `gpio:gpio_value`
- Kprobes
- Histogram trigger
- Synthetic event
- `trace_printk()`
- Ftrace plugin `function`
- Ftrace plugin `function_graph`

→ Most tracing facilities can be combined with filters and / or triggers

Tracing facilities for userspace

- Tracing event `syscalls:sys_enter_ioctl` and `syscalls:sys_exit_ioctl`
- Uprobes
- Trace marker
- `strace`

Testsystem BeagleBone Black

- TI-AM3358 (BeagleBone Black) with 1000 MHz
- linux-6.1.26 with preemption and linux-6.1.26-rt8 with rt patch
- Performance optimized kernel configuration
- No worst case values measured

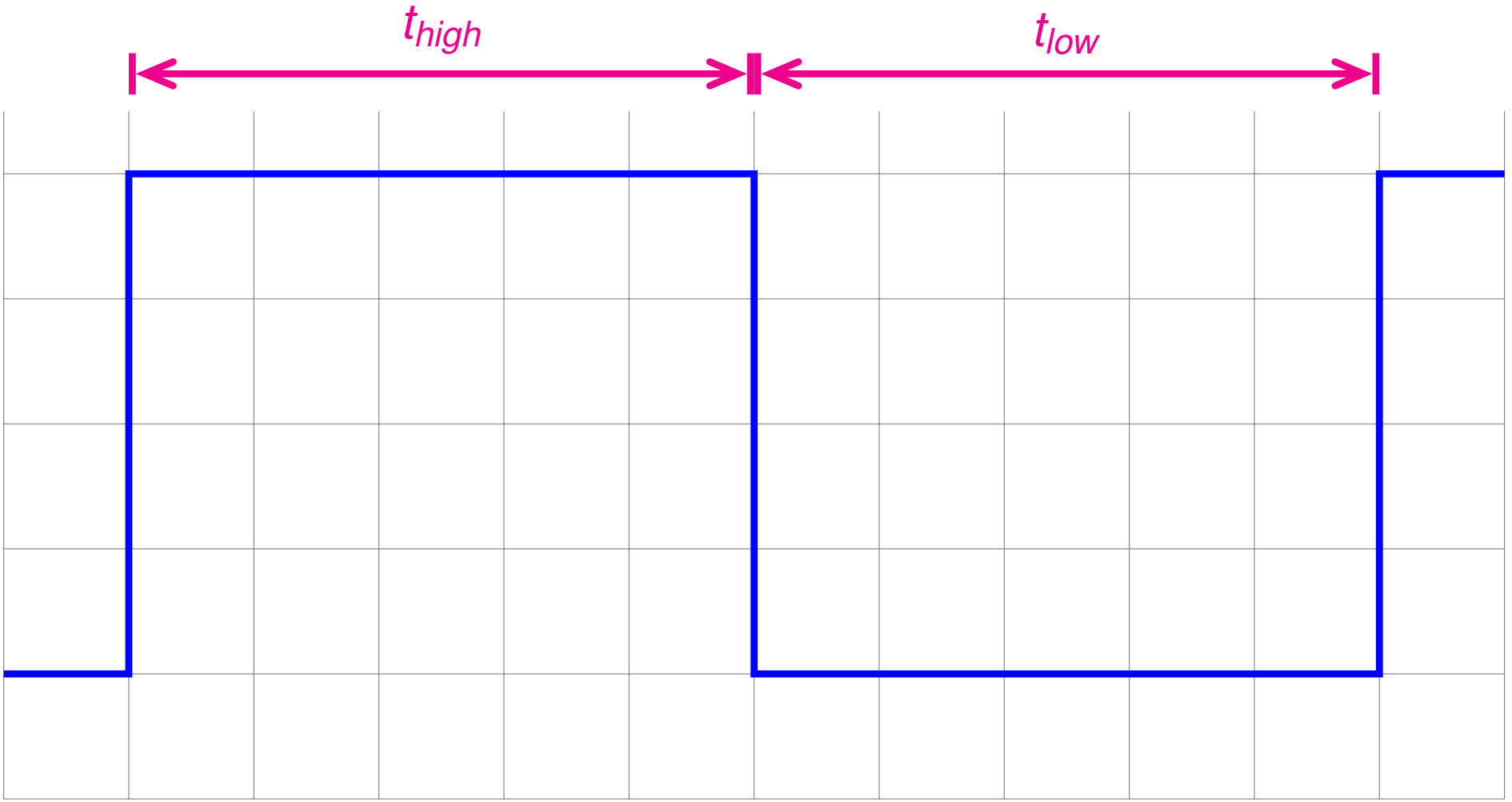
Toggeling of the GPIO in Driver

```
struct toggle_data {
        struct gpio_desc        *gpiod_toggle;
        ...
};
void toggle_once(struct toggle_data *data)
{
        gpiod_set_value(data->gpiod_toggle, 1);
        gpiod_set_value(data->gpiod_toggle, 0);
}
int toggle_gpio_sleep(void *arg)
{
        int i;
        struct toggle_data *data = (struct toggle_data*)arg;
        while (1) {
                for (i = 0; i < 5; i++)
                        toggle_once(data);
                if (kthread_should_stop())
                        break;
                msleep(1);
        }
        return 0;
}
```

Toggeling in kernel driver

- Function `toggle_gpio_sleep()` runs as realtime kernel thread
- `msleep()` is used for avoiding realtime throttling
- Measuring on the oscilloscope on the second call to `toggle_once()` of one phase after the `msleep()`
  $\rightarrow$ Measuring cache hot scenario
- Each measuring is done with a minimum of 30000 wave counts
  $\rightarrow$ Random samples with more than 500000 wave counts verified that measured average values and standard deviation are correct

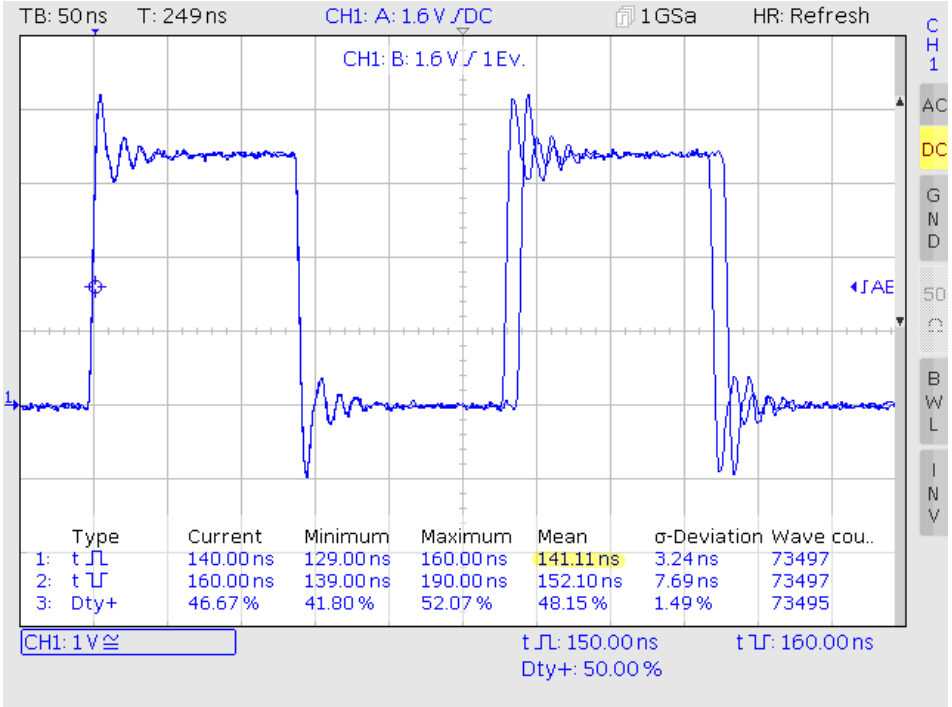
Measuring with the oscilloscope



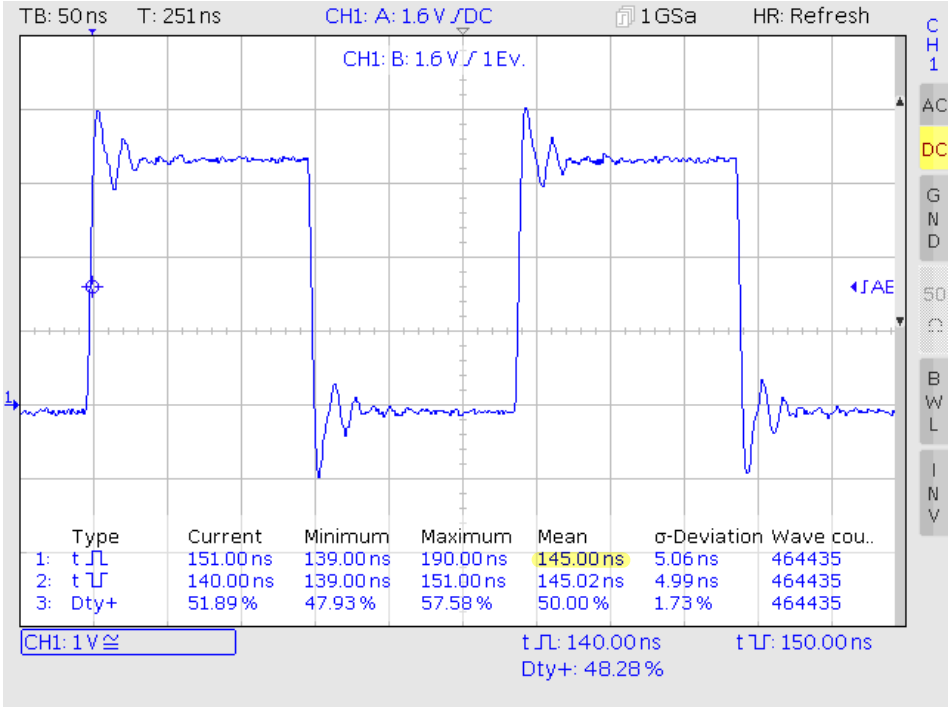
$$Dty_{+} = \frac{t_{high}}{t_{high}+t_{low}}$$

Realtime preemption patch

Minimal configuration with preemption (v6.1.26)



Minimal configuration with rt patch (v6.1.26-rt8)



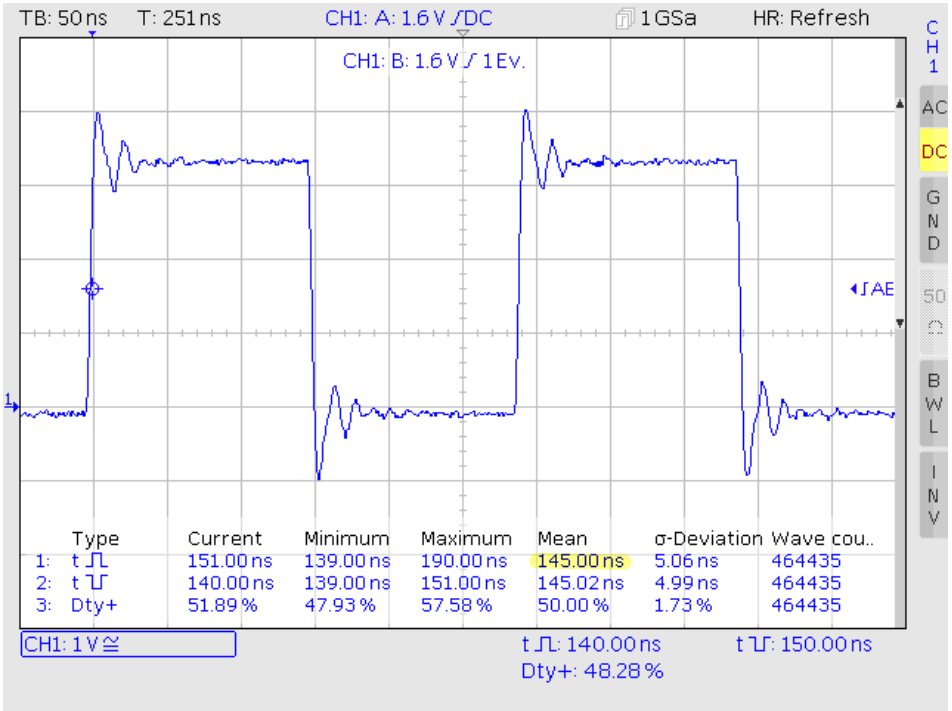
⇒ *no significant difference in time response*

Difference in configuration:

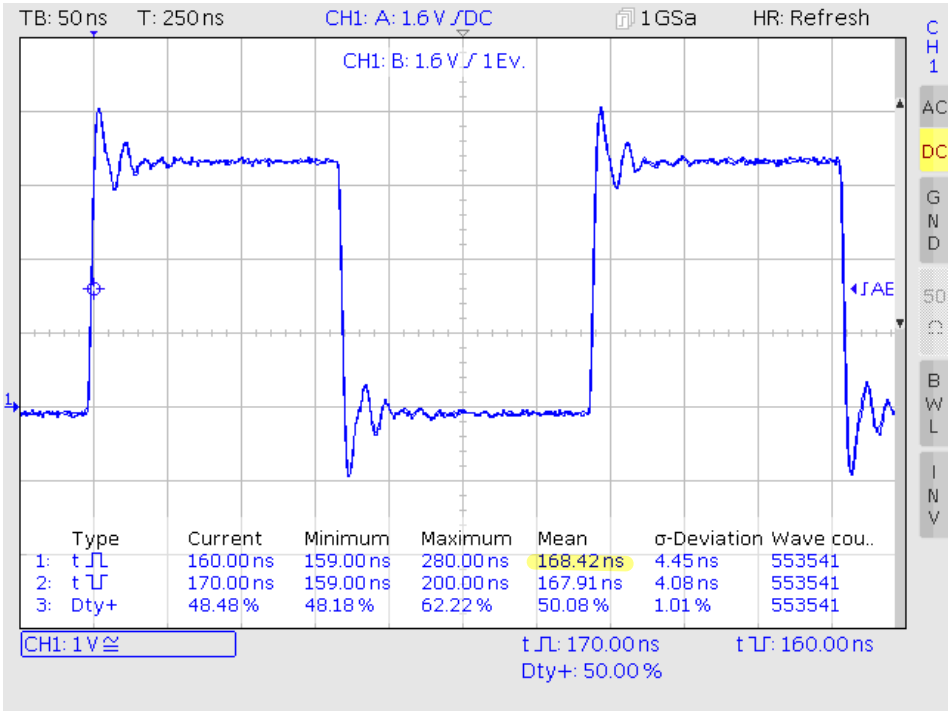
```
$ diff bbb_min_defconfig bbb_rt_min_defconfig
9c9
< CONFIG_PREEMPT=y
---
> CONFIG_PREEMPT_RT=y
```

Tracing infrastructure

Minimal configuration with rt patch (v6.1.26-rt8)



Tracing configured with rt patch (v6.1.26-rt8)



→ *Tracing configuration overhead:* 23*ns* (16%)

Overhead with v6.1.26: (scope dump not shown)
→ 26*ns* (19%)

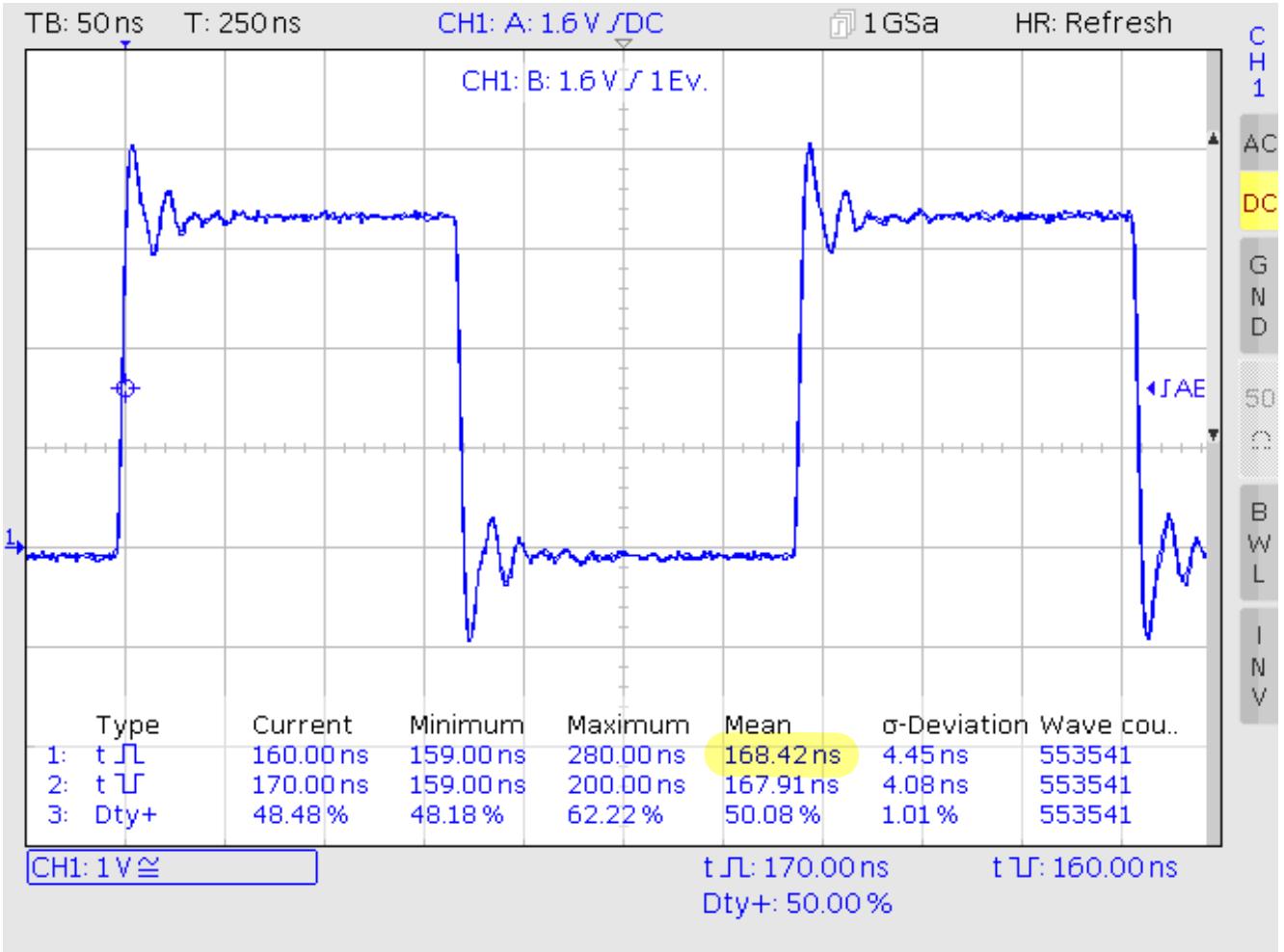
Differences in configuration

```
$ diff bbb_min_defconfig bbb_trace_defconfig
1c1
< CONFIG_LOCALVERSION="-bbb-min"
---
> CONFIG_LOCALVERSION="-bbb-trace"
221a222
> CONFIG_DEBUG_FS=y
226c227,234
< # CONFIG_FTRACE is not set
---
> CONFIG_STACK_TRACER=y
> CONFIG_TIMERLAT_TRACER=y
> CONFIG_FTRACE_SYSCALLS=y
> CONFIG_HIST_TRIGGERS=y
> CONFIG_TRACEPOINT_BENCHMARK=y
> CONFIG_HIST_TRIGGERS_DEBUG=y
> CONFIG_BACKTRACE_VERBOSE=y
> CONFIG_DEBUG_USER=y
```

Results from tracing infrastructure

Pulse width +	**v6.1.26**	**v6.1.26-rt8**	Δ
no tracing	141.11 ns (SD = 3.24)	145.00 ns (SD = 5.06)	*+3.89 ns +2.76 %*
with tracing	167.30 ns (SD = 4.40)	168.42 ns (SD = 4.45)	*+1.12 ns +0.67 %*
Δ	*+26.19 ns +18.56 %*	*+23.42 ns +16.15 %*	

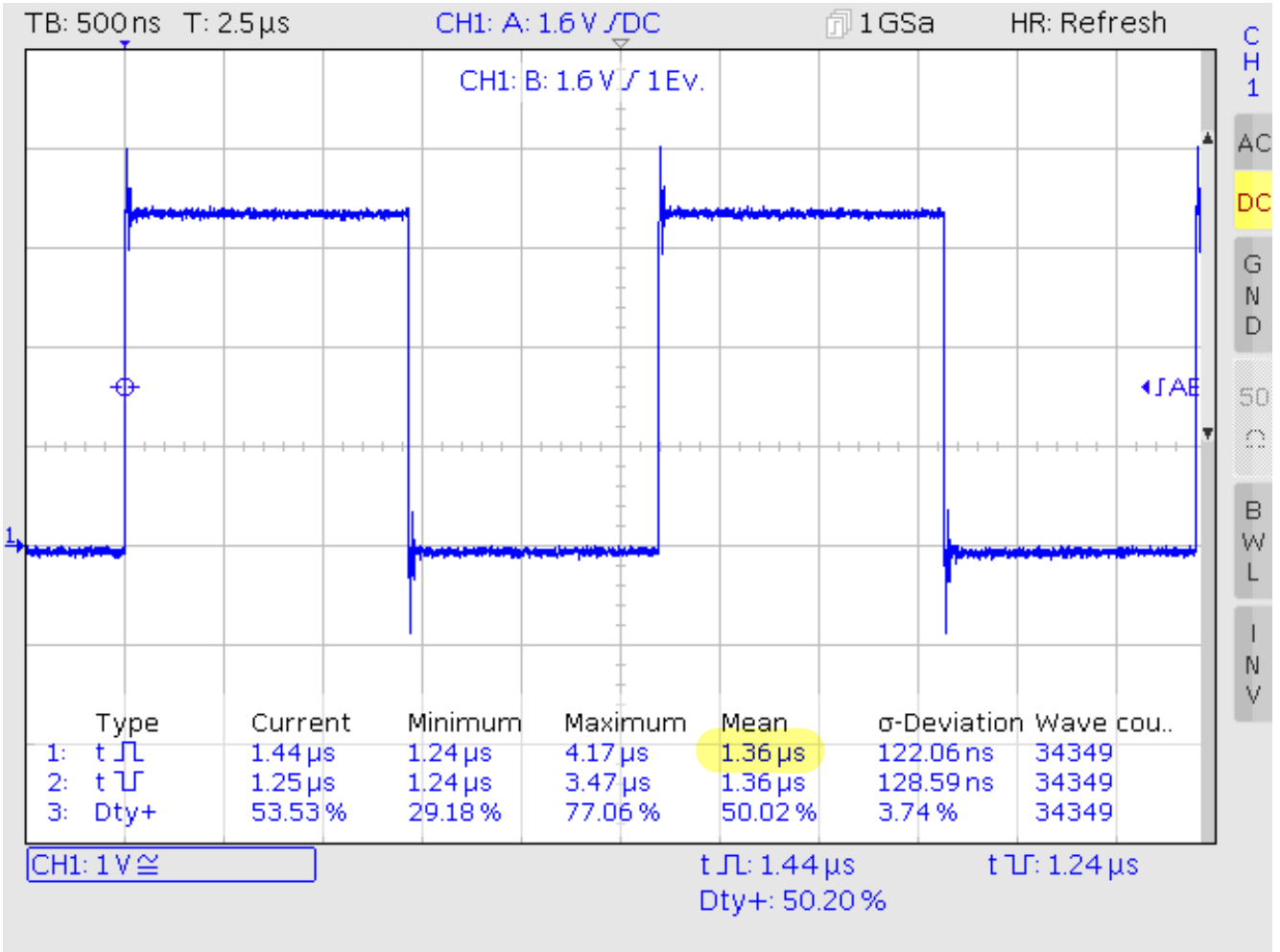
Reference configuration for tracing overhead



- BeagleBone Black with vanilla kernel and rt preemption patch (v6.1.26-rt8)
- `CONFIG_PREEMPT_RT` and tracing configured (`bbb_trace_defconfig`) but not activated

```
$ insmod toggle.ko mode=10
```

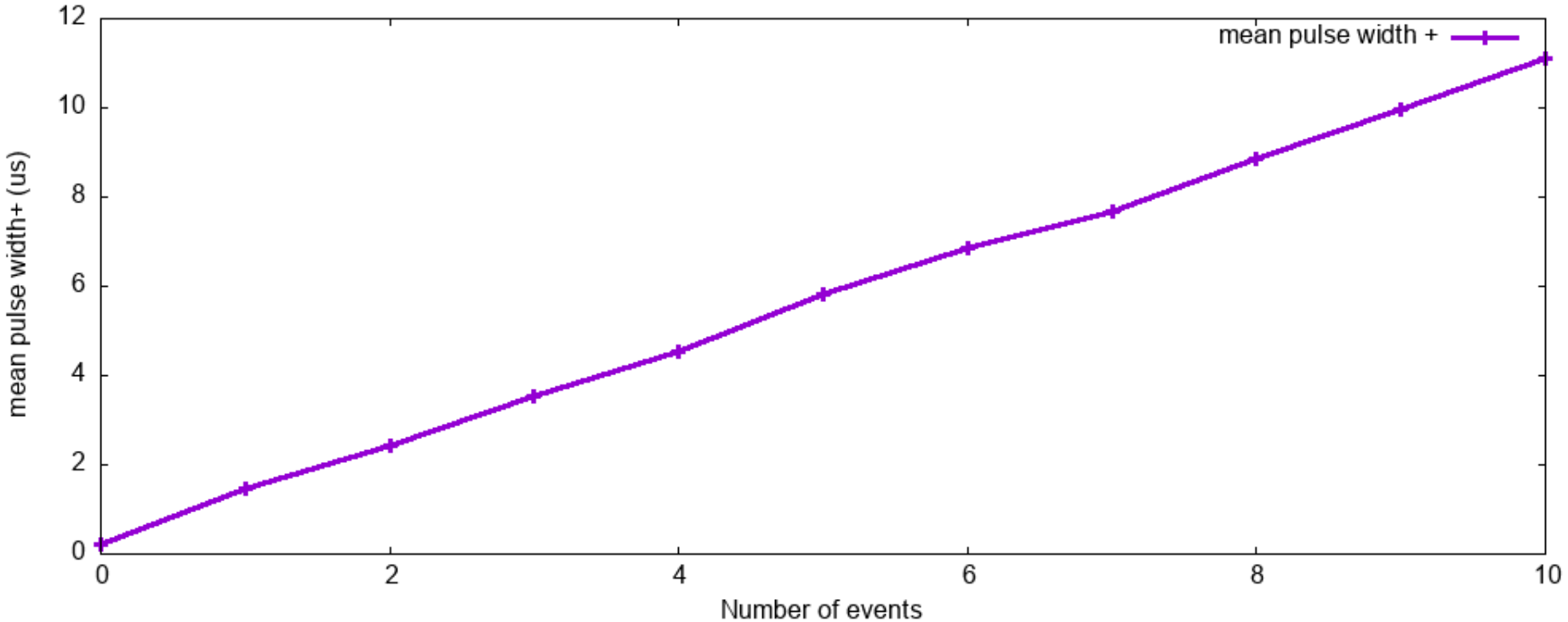
Activating an event



→ *Tracing event overhead:* 1.19μs (709%)

```
$ echo 1 > /sys/kernel/tracing/events/gpio/gpio_value/enable
$ cat /sys/kernel/tracing/trace
```

Dependency on number of events



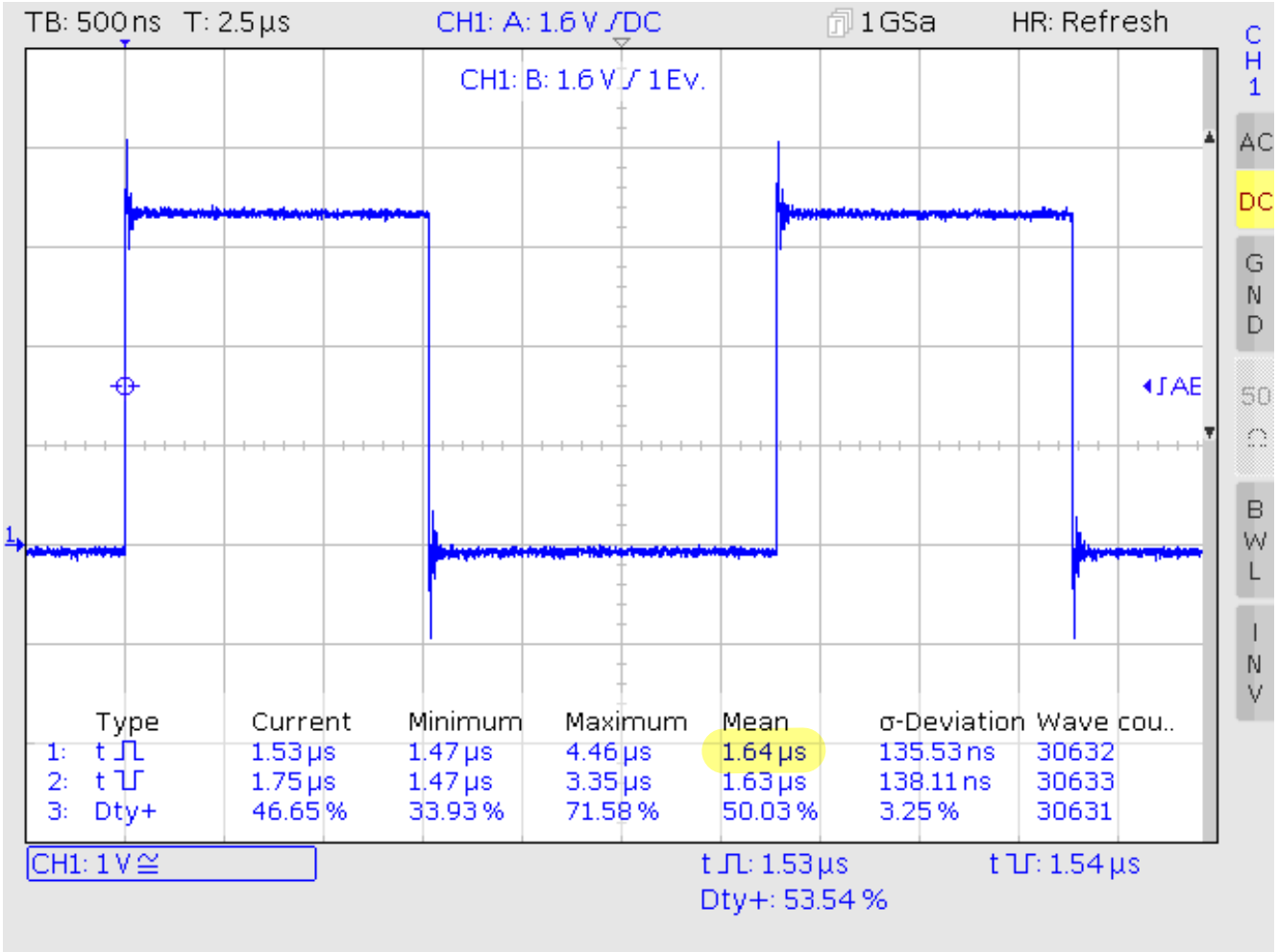
Measuring with a special driver variant with 10 events on each gpiod_set_value() which can be switched on and off individually. Each event monitors two u32 values:

number of events	0	1	2	3	4	5	10
pulse width+ (μs)	0.23	1.46	2.44	3.53	4.53	5.82	11.1

$\rightarrow$ Overhead increases linear with number of events.

$\Rightarrow$ Each additional event costs about 1.1μs.

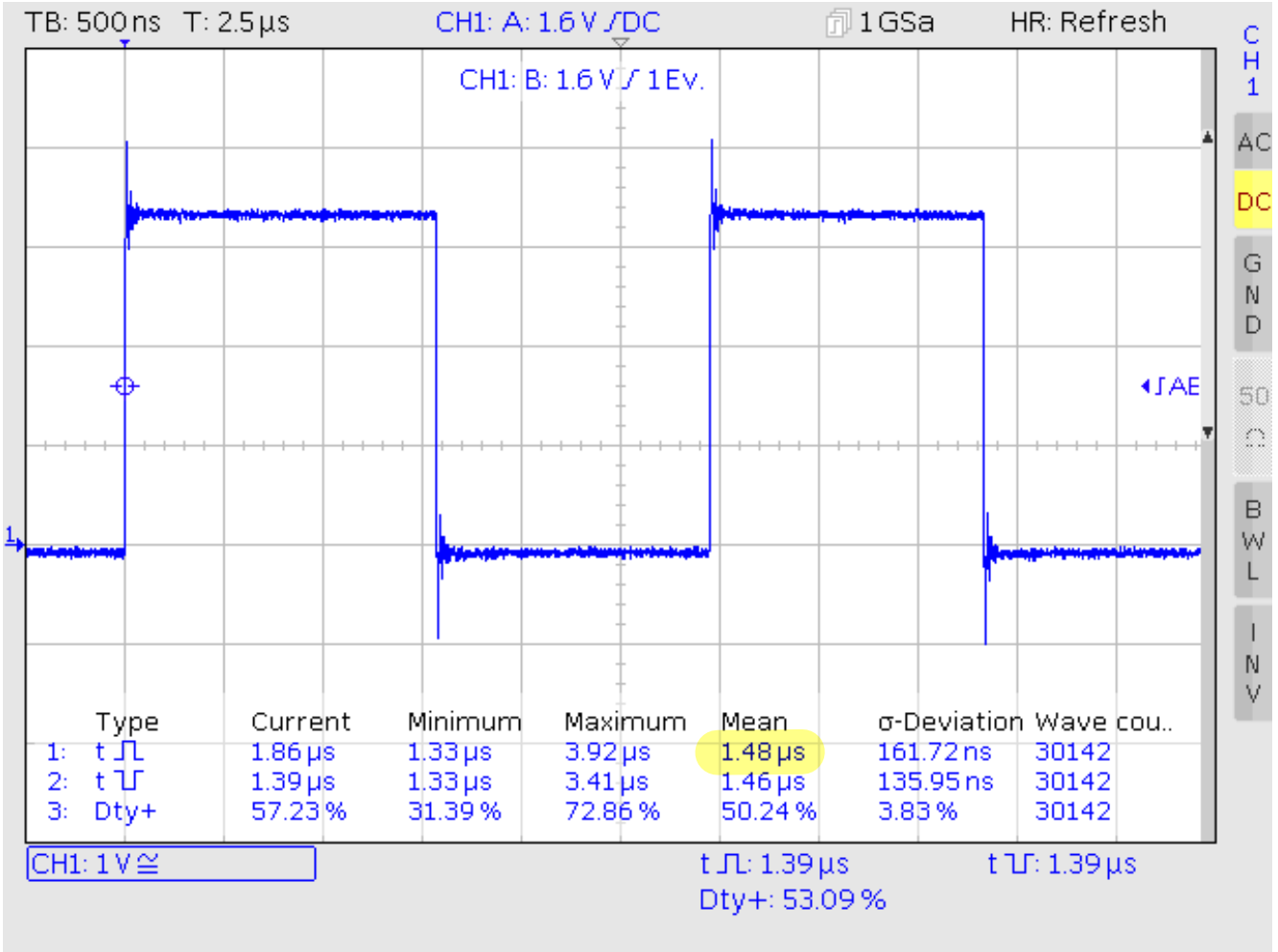
Event with filter



→ *Filter overhead:* 0.28μs (21%) *(compared to tracing event)*

```
$ echo 1 > /sys/kernel/tracing/events/gpio/gpio_value/enable
$ echo 'gpio == 70' > /sys/kernel/tracing/events/gpio/gpio_value/filter
$ cat /sys/kernel/tracing/trace
```

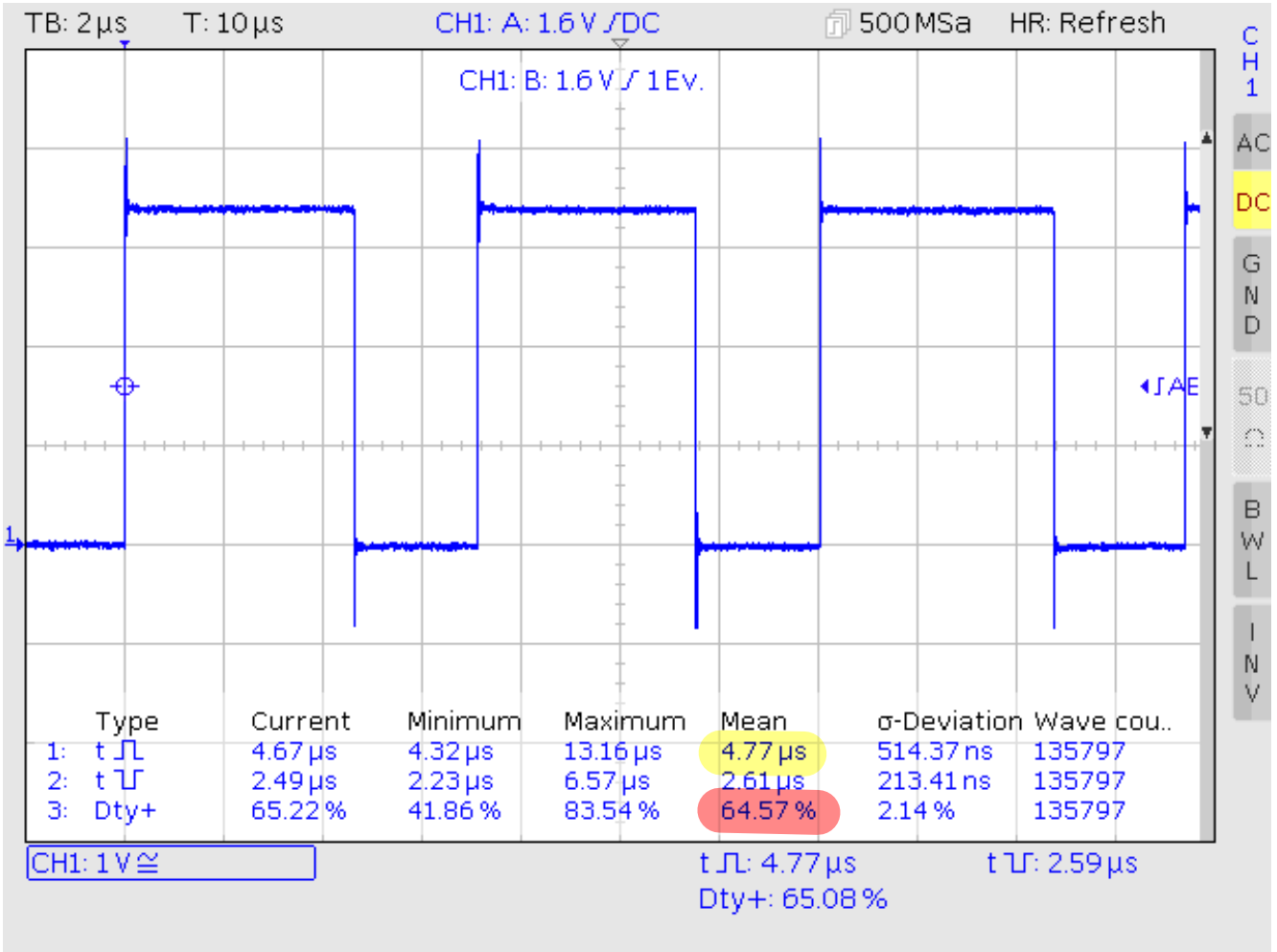
Event with trigger



→ *Trigger overhead:* 0.12μs (9%) *(compared to tracing event)*

```
$ echo 1 > /sys/kernel/tracing/events/gpio/gpio_value/enable
$ echo 'traceon' > /sys/kernel/tracing/events/gpio/gpio_value/trigger
$ cat /sys/kernel/tracing/trace
```

Event with hist trigger



→ *Hist trigger overhead:* 3.41 μs (251%) *(compared to tracing event)*

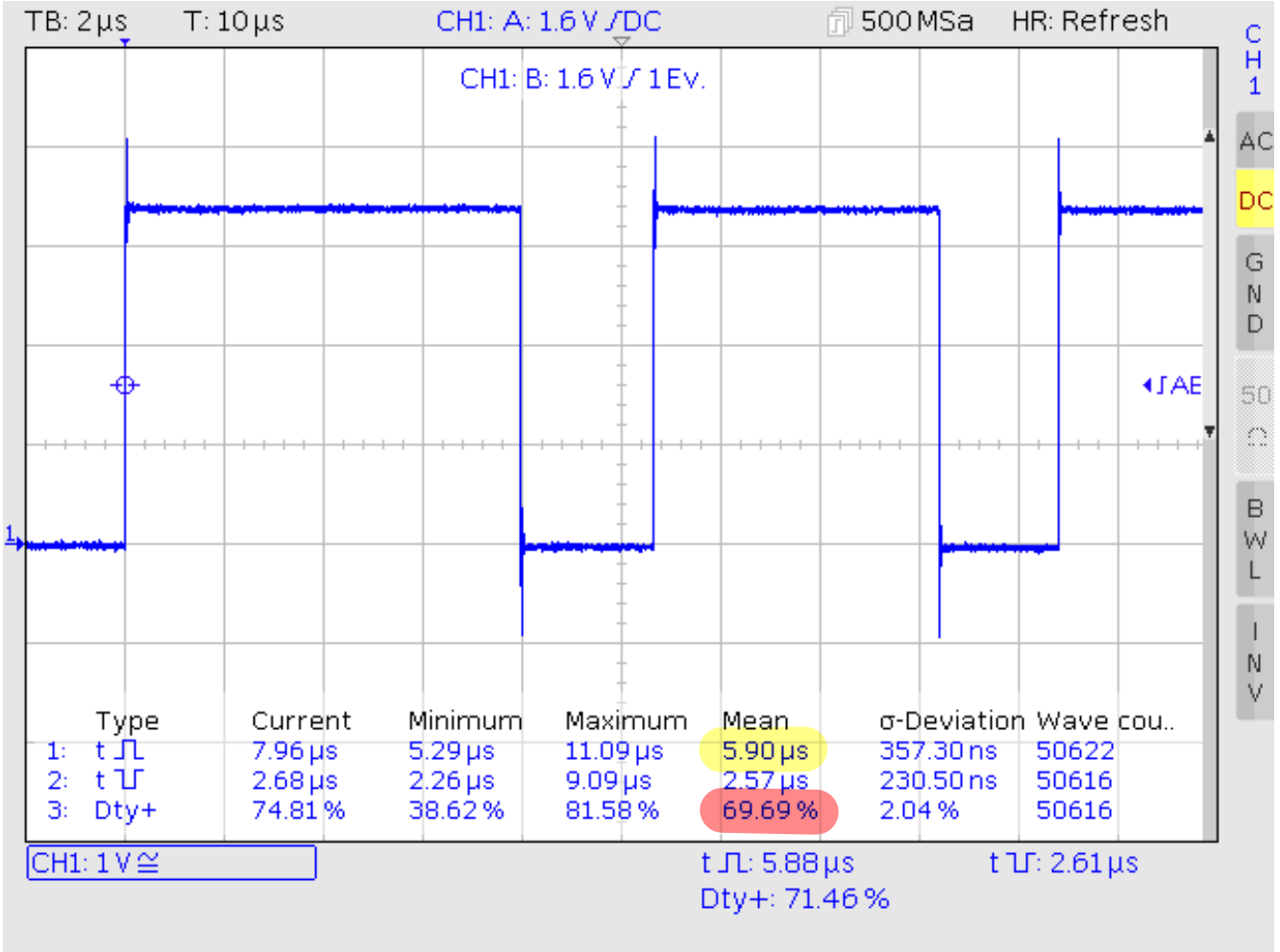
Why is Dty_+ to high?

→ not yet clarified

Event with hist trigger

```
$ echo 'gpio_high u32 gpio u64 lat'                                    \
       >> /sys/kernel/tracing/synthetic_events

$ cd /sys/kernel/tracing/events/gpio/gpio_value
$ echo 1 > enable

$ echo 'hist:keys=gpio:ts0=common_timestamp if value == 1' >> trigger

$ echo 'hist:keys=gpio:lat=common_timestamp-$ts0:                      \
       onmatch(gpio.gpio_value).trace(gpio_high,gpio,$lat)             \
       if value == 0' >> trigger

$ cd /sys/kernel/tracing/events/synthetic/gpio_high
$ echo 1 > enable

$ cat /sys/kernel/tracing/trace
toggle-gpio-sle-125  [000] 84892.334011: gpio_high: gpio=70 lat=2459
toggle-gpio-sle-125  [000] 84892.334014: gpio_high: gpio=70 lat=2333
```

Synthetic event on hist trigger

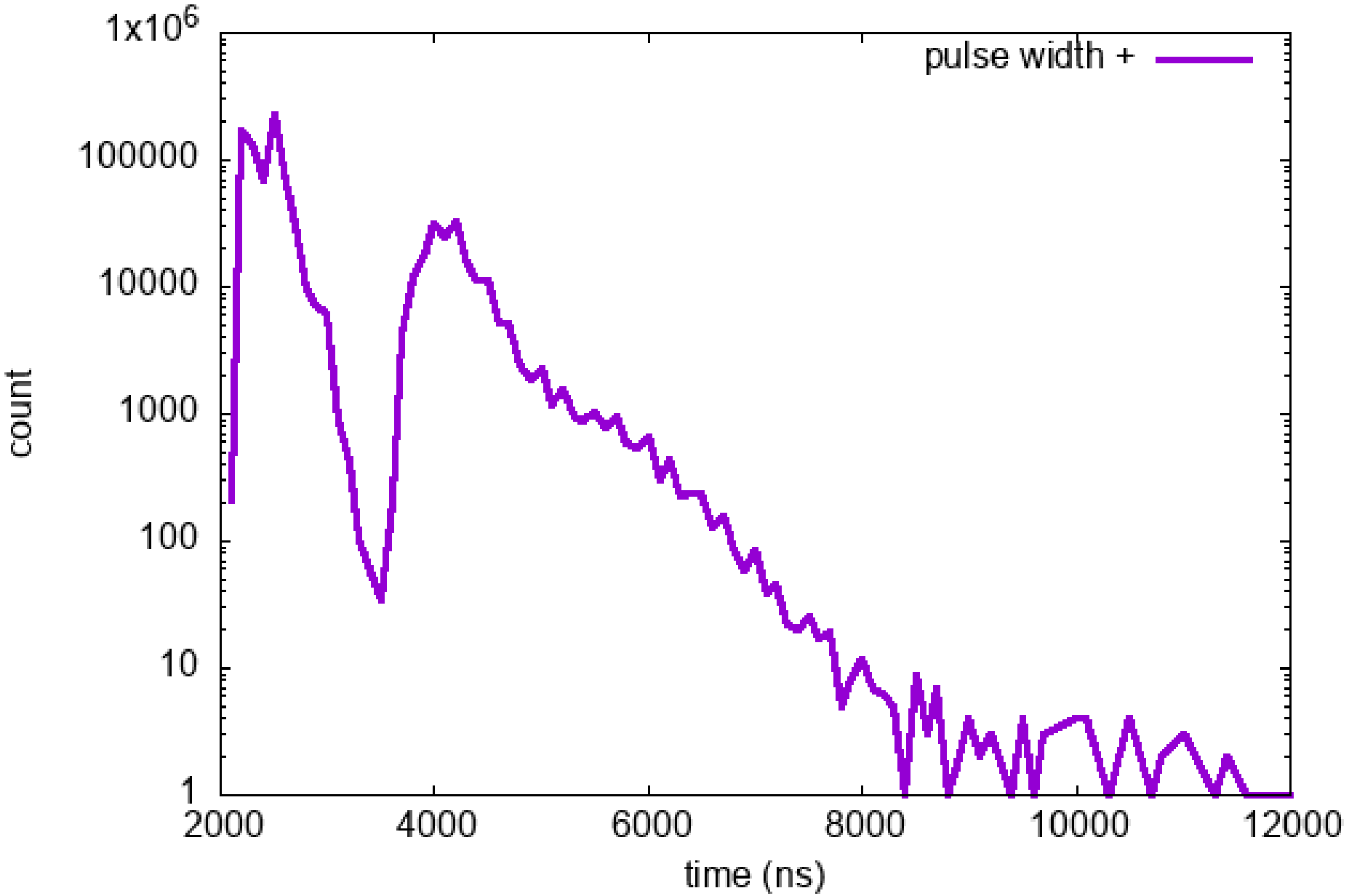


→ *Synthetic (plus hist) trigger overhead:* 4.54μs (334%) *(compared to tracing event)*

Synthetic event on hist trigger

```
# everything from hist trigger plus:

$ cd /sys/kernel/tracing/events/synthetic/gpio_high
$ echo 'hist:key=gpio,lat.buckets=100:sort=lat' >> trigger

$ cat hist
{ gpio:          70, lat: ~ 2000-2099 } hitcount:       1766
{ gpio:          70, lat: ~ 2100-2199 } hitcount:      99393
{ gpio:          70, lat: ~ 2200-2299 } hitcount:     130325
{ gpio:          70, lat: ~ 2300-2399 } hitcount:      56458
{ gpio:          70, lat: ~ 2400-2499 } hitcount:      99428
...
```

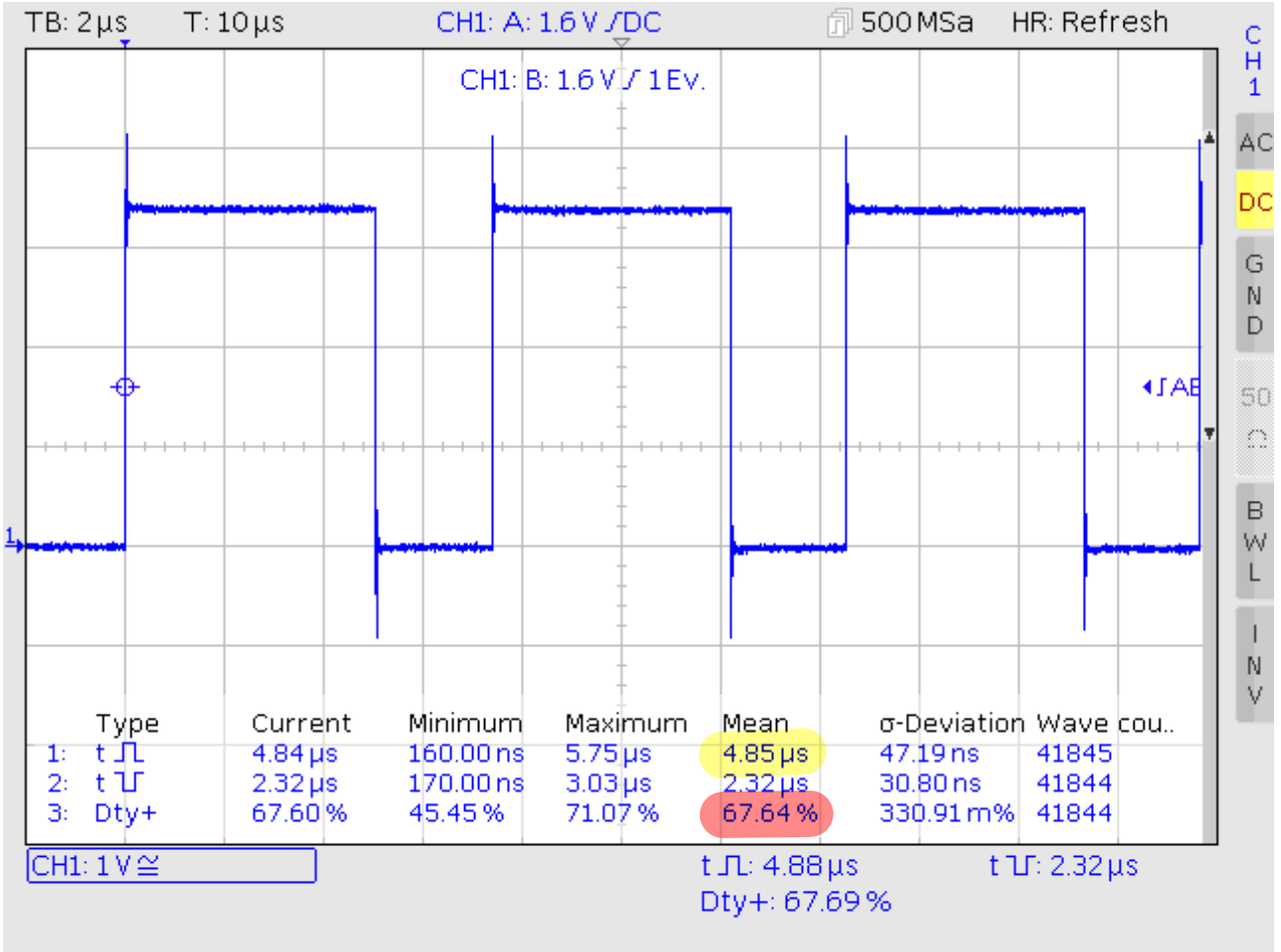
Synthetic event on hist trigger



Synthetic event on hist trigger

```
$ cd /sys/kernel/tracing/events/synthetic/gpio_high
$ echo 'hist:key=gpio,lat.buckets=100:sort=lat' >> trigger

$ cat hist > ~/synthetic.txt

$ cat synthetic.txt | awk 'BEGIN {FS="-"} {print $1 "\t" $2}' |
    awk '{print $6 "\t" $10}' | grep -v '^[[:space:]]$' > syn-table.txt

$ gnuplot -e "file='syn-table.txt'"  \
          -e "pngfile='synthetic-table.png'" synt-hist.gp

$ cat synthetic-table.txt |
    awk '{n+=$2; x+=($1+50)*$2} END{print x/n " ns/pulse+"}'
2706.49 ns/pulse+

# the same with rt kernel delivers:
2851.16 ns/pulse+
```

Histogram with eBPF and ply

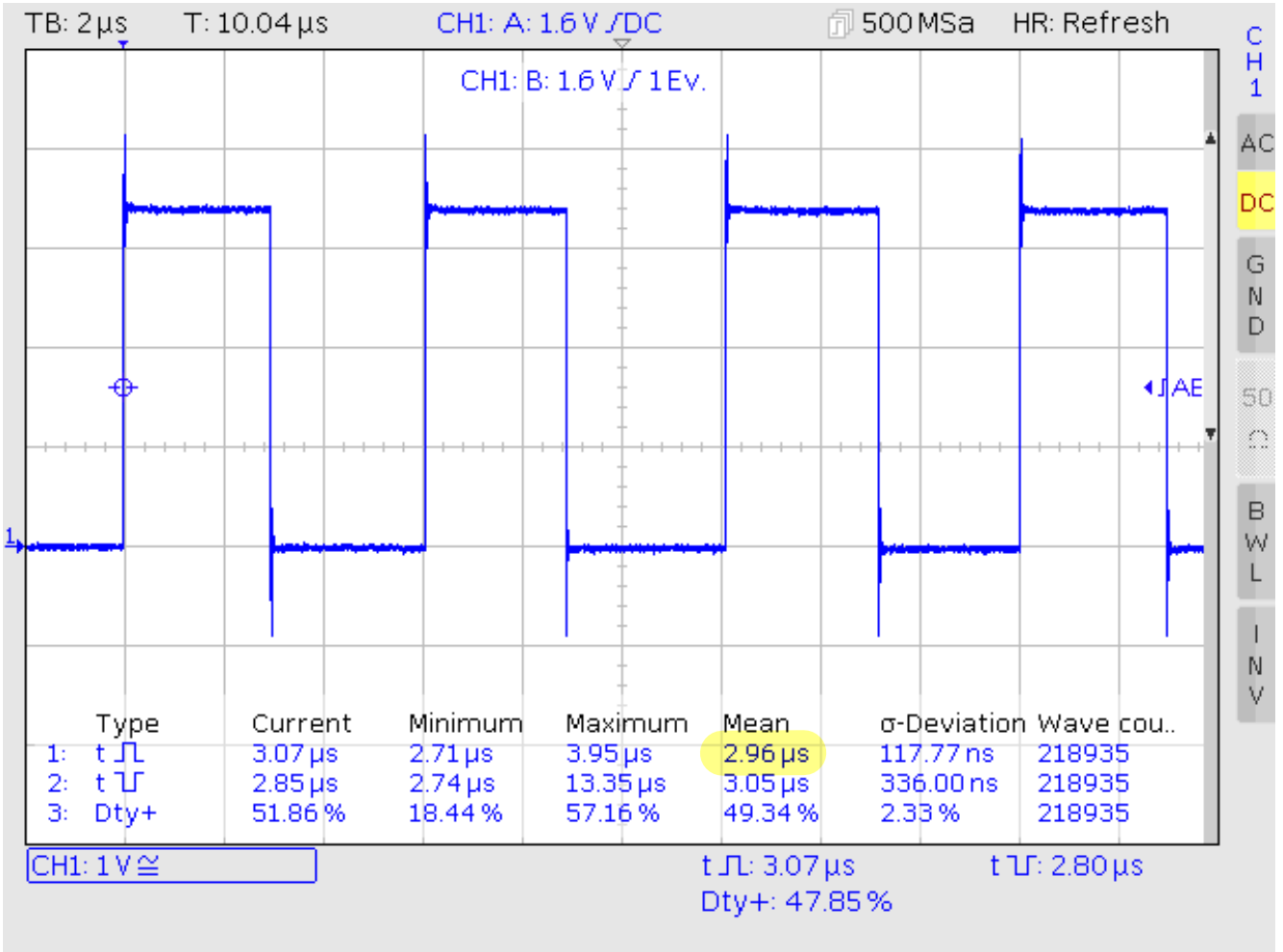


→ *Histogram with ply overhead:* 3.49μs (257%) *(compared to tracing event)*

Histogram with eBPF and ply

```
$ cat delta.ply
tracepoint:gpio/gpio_value {
    if (data->value == 1)
        start[data->gpio] = time;
    else
        @delta[data->gpio] = quantize(time - start[data->gpio]);
    }
}

$ ply delta.ply
...
@delta:
{ 70 }:
  [2.04us, 4.09us] 616075 |#########################      |
  [4.09us, 8.19us] 141473 |#####                          |
  [8.19us, 16.3us]      5 |                               |
  [16.3us, 32.7us]      2 |                               |
```

One kprobe with ftrace

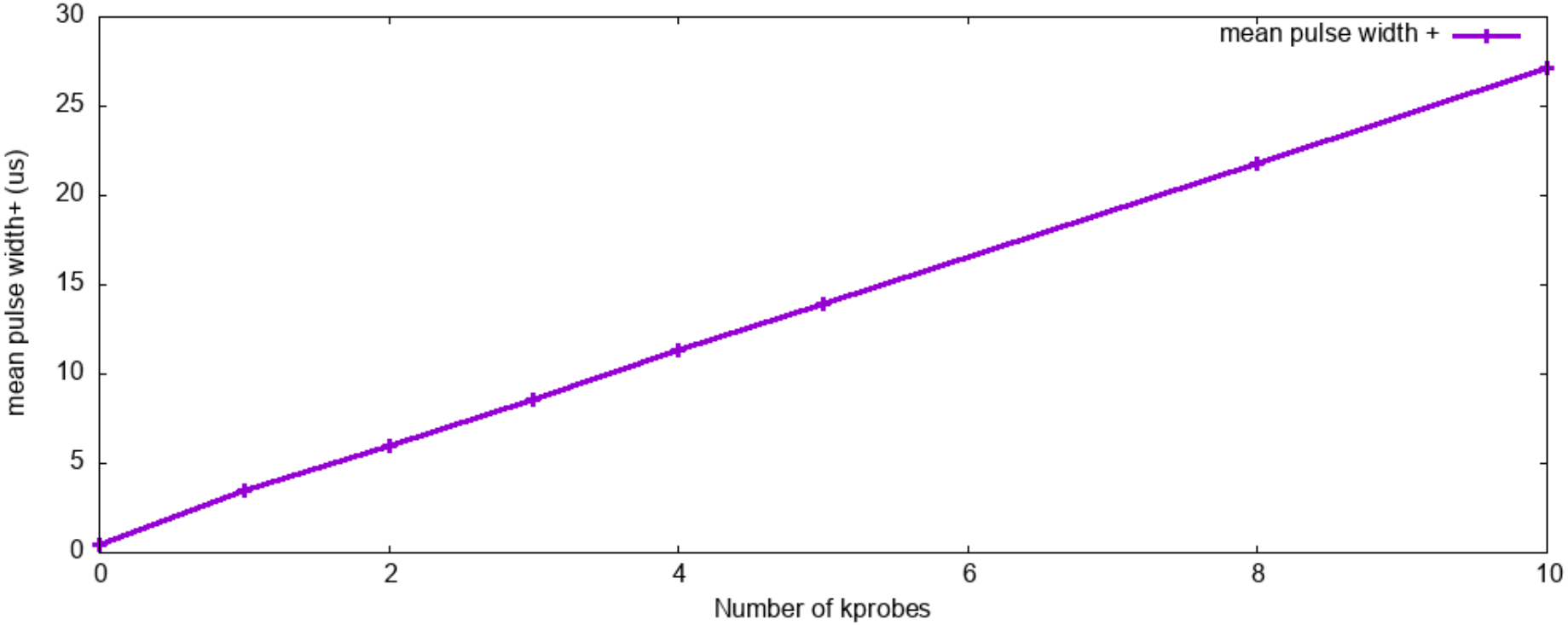


→ *kprobe overhead:* 2.79μs *(compared to tracing event)*

kprobe with ftrace

```
$ cd /sys/kernel/tracing/
$ echo 'p:gpiod_set_value gpiod_set_value \
       name=+0(+12(%r0)):string value=%r1:u32' > kprobe_events

$ cd /sys/kernel/tracing/events/kprobes
$ echo 1 > gpiod_set_value/enable

$ cat /sys/kernel/tracing/trace
toggle-gpio-sle-139 [000] 7666.934324: gpiod_set_value:
            (gpiod_set_value+0x0/0xc4) name="P8_45 [hdmi]" value=1
...
```

kprobe with ftrace



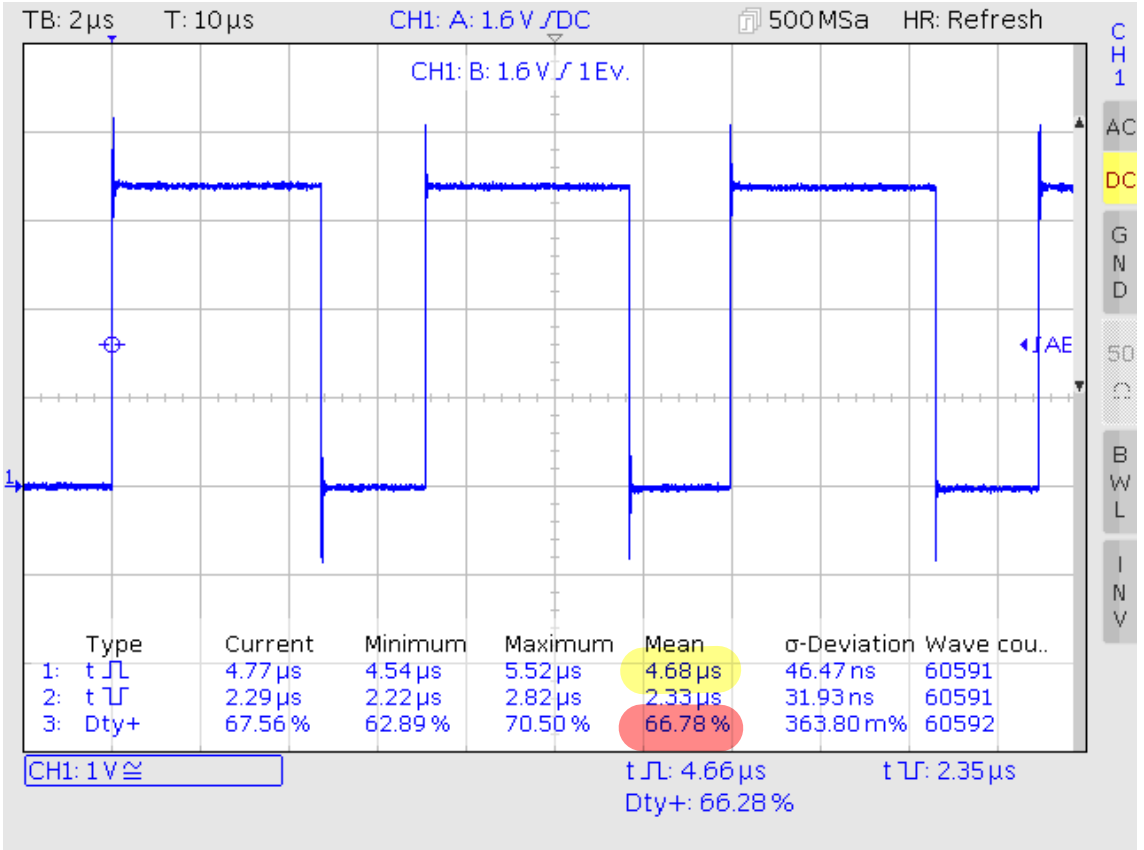
Measuring with a special driver variant with 10 function calls (without optimizer) until gpipd_set_value is called:

number of kprobes	0	1	2	3	4	5	10
pulse width+ (μs)	0.46	3.45	5.98	8.57	11.3	13.9	27.1

$\rightarrow$ Overhead increases linear with number of kprobes.

$\Rightarrow$ Each additional kprobe costs about 2.6μs.

kprobe with eBPF




```
$ cat dt-kprobe.ply
kprobe:gpiod_set_value {
    if (arg1 == 1) {
        start[0] = time;
    } else {
        @delta[0] = quantize(time - start[0]);
    }
}
```

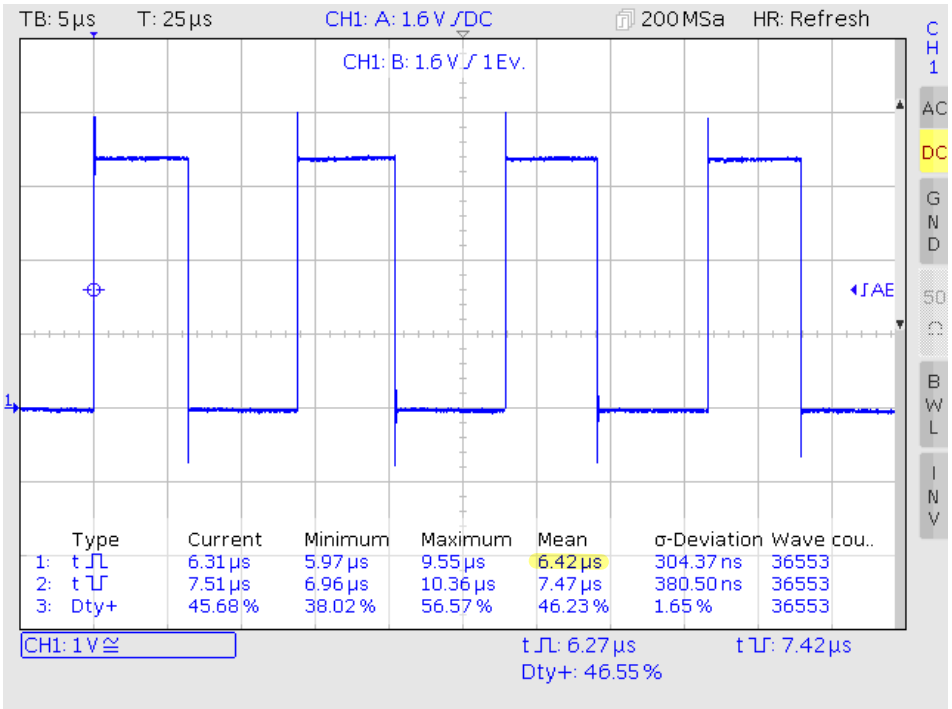
kprobe with eBPF

with rt kernel:

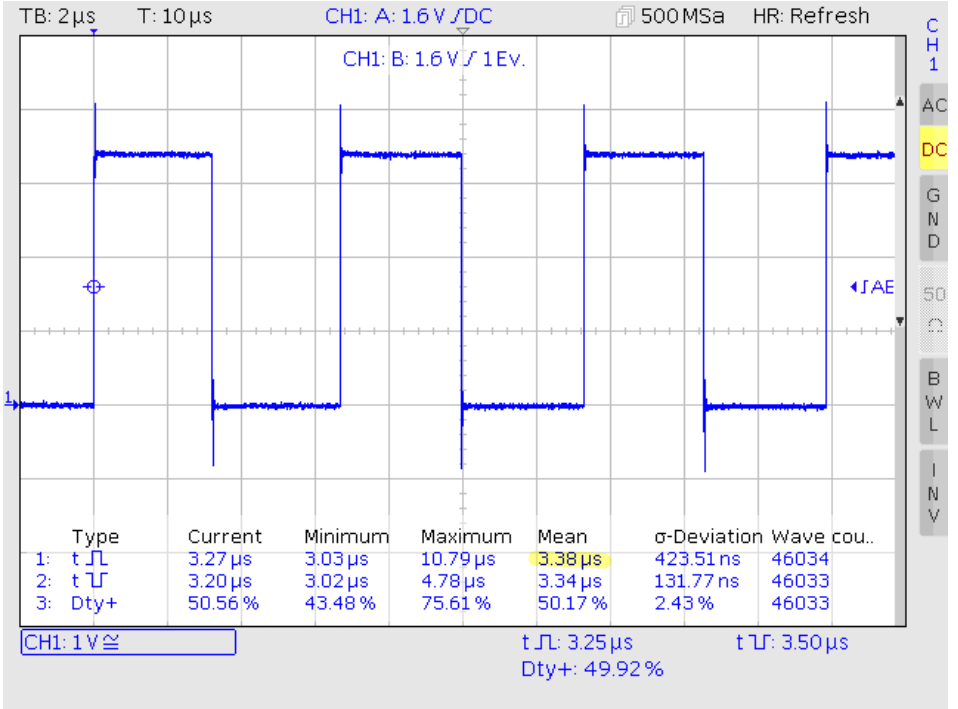
```
$ ply dt-kprobe.ply
...
@delta:
   [2.04us, 4.09us] 885090 |############################      |
   [4.09us, 8.19us] 215506 |#######                           |
   [8.19us, 16.3us]      7 |                                   |
   [16.3us, 32.7us]      2 |                                   |
```

Plugin function

no filtering



with set_ftrace_filter



→ Pulse width changes when using filters by `set_ftrace_filter`

⇒ Overhead can partly be reduced with filters

→ *function plugin overhead with 3 functions traced:* $3.21\,\mu s$ *(compared to untraced)*

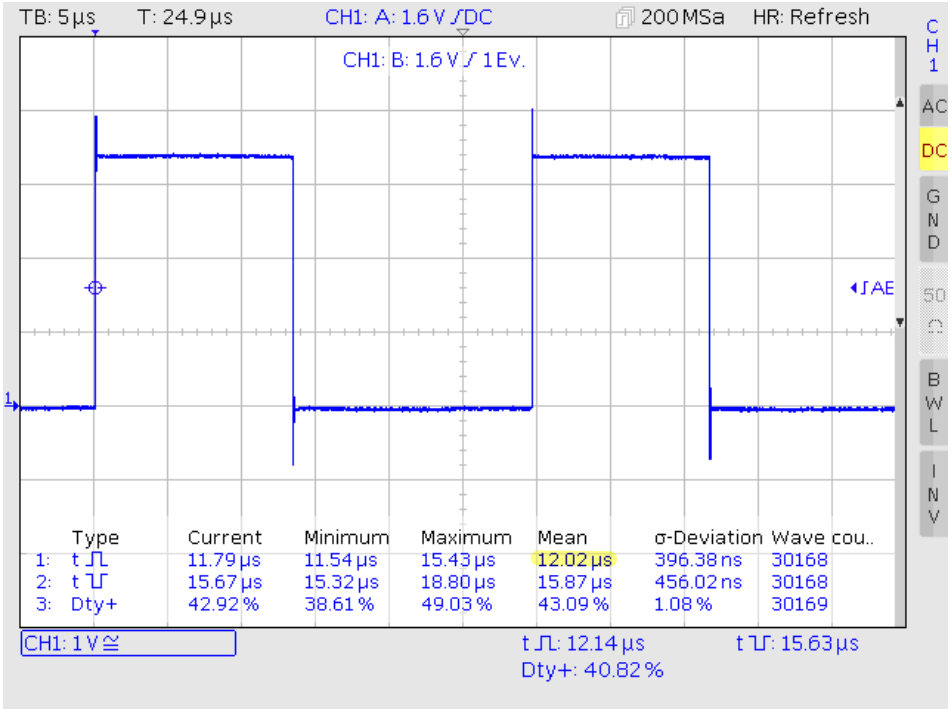
Plugin function

```
$ cd /sys/kernel/tracing/
$ echo function > current_tracer

$ echo 'gpiod*' > set_ftrace_filter
5
$ cat trace
toggle-gpio-sle-139 [000]   615.644329: gpiod_set_value <-toggle_once
toggle-gpio-sle-139 [000]   615.644341: gpiod_set_value_nocheck ...
toggle-gpio-sle-139 [000]   615.644342: gpiod_set_raw_value_commit ...
```

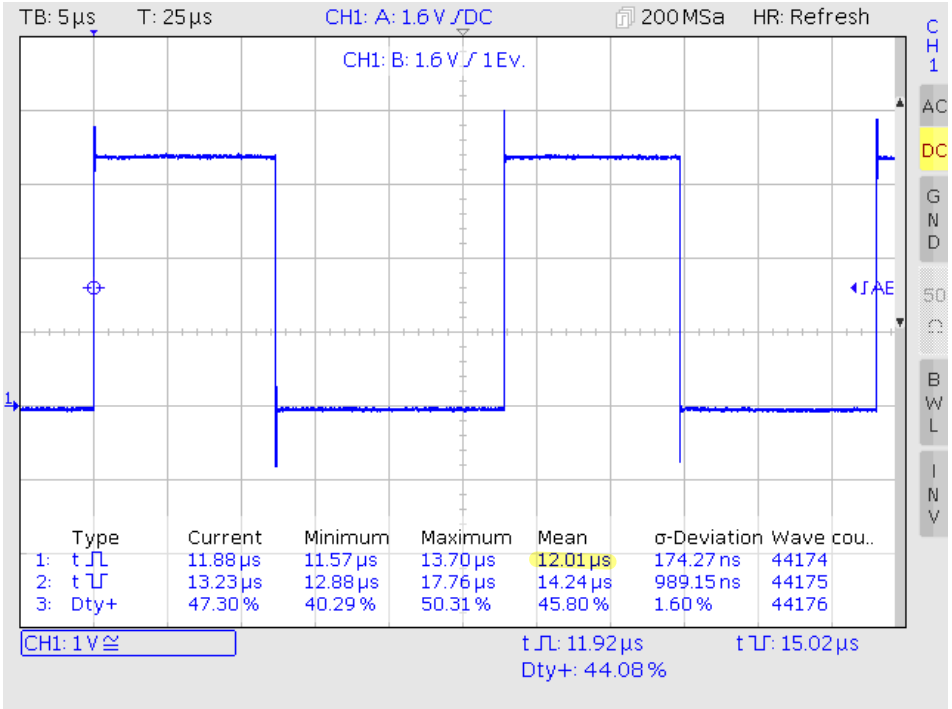
$\Rightarrow$ 3 tracing outputs for one gpio edge

Plugin function_graph

no filtering



with set_graph_function



→ *function_graph overhead:* 11.84μs (70 *times higher*) *(compared to untraced)*

Plugin function_graph

- Pulse width doesn't change much when reducing the trace volume by using the `set_graph_function`
  $\Rightarrow$ Overhead is only slightly dependent on number of traced functions
- Time values in tracing output are much higher than in productive kernel

Plugin function_graph

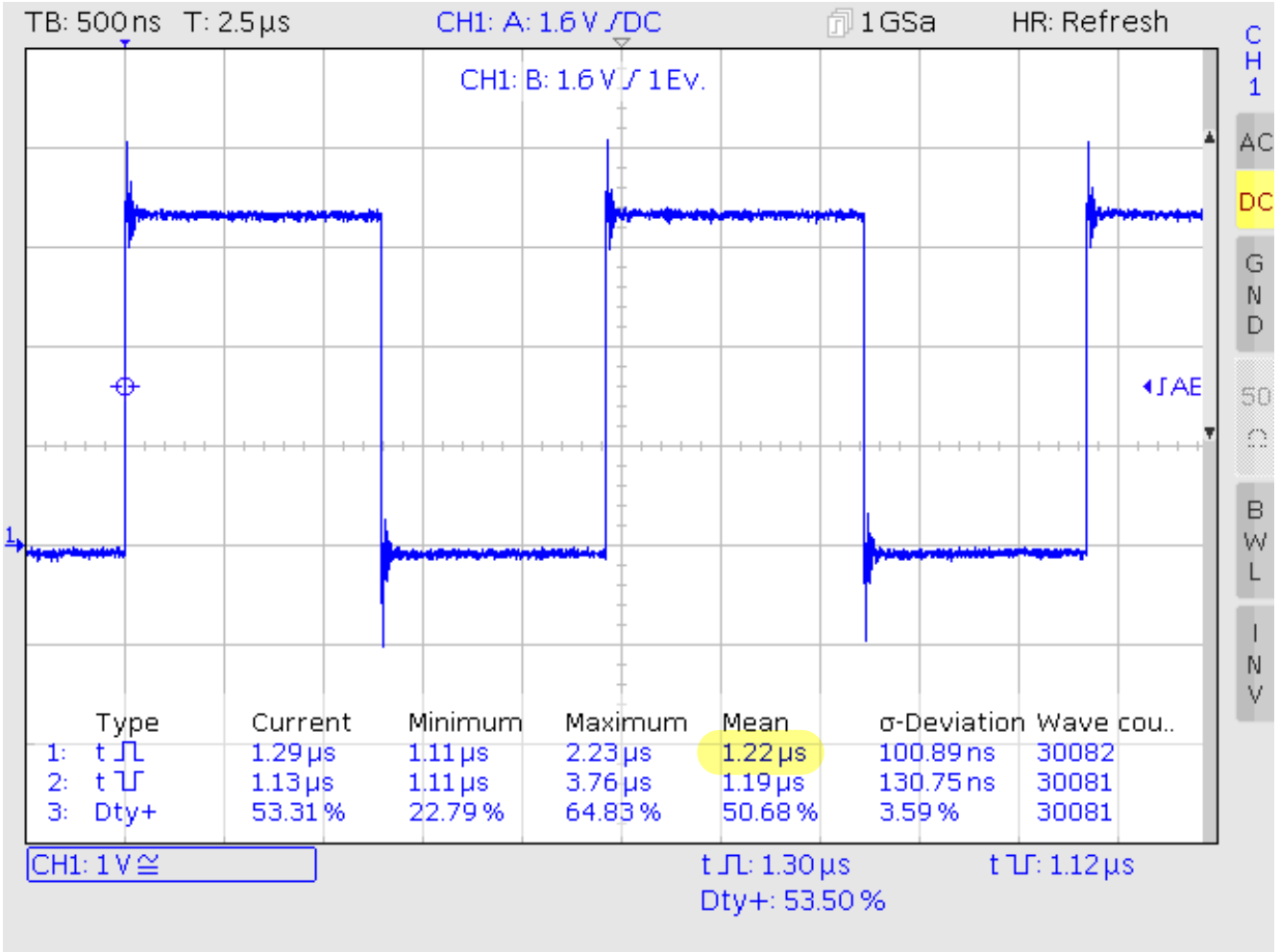
```
$ cd /sys/kernel/tracing/
$ echo function_graph > current_tracer
$ echo gpiod_set_value > set_graph_function

$ cat trace
 0)               |  gpiod_set_value() {
 0)               |    gpiod_set_raw_value_commit() {
 0)    6.000 us   |      omap_gpio_set();
 0) + 10.125 us   |    }
 0) + 16.292 us   |  }
```

Calculation:

$\frac{16292}{70} = 233ns$

$\rightarrow$ close to what was measured without tracing (168*ns*).

trace_printk()

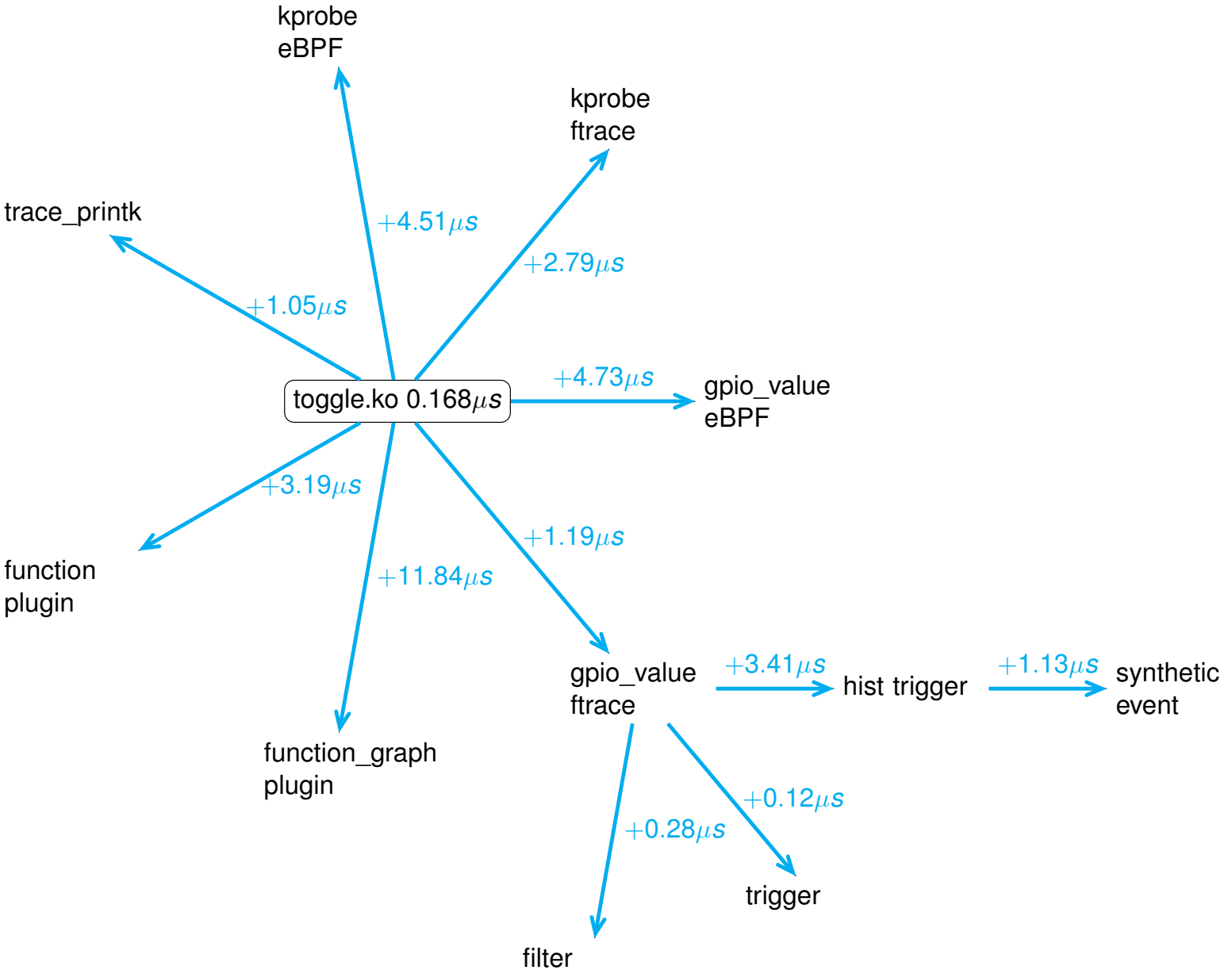


→ *trace_printk overhead:* 1.05μs (626%) *(compared to untraced)*

trace_printk()

```
static int toggle_trace_printk(void *arg)
{
    struct toggle_data *data = (struct toggle_data*)arg;
    int i = 0;
    while (1) {
        trace_printk("gpio-toggle\n");
        gpiod_set_value(data->gpiod_toggle, i++ & 0x01);
        if ((i & 0xFFFFF) == 0xFFFFF && kthread_should_stop())
            break;
    }
    return 0;
}
```

Tracing overhead



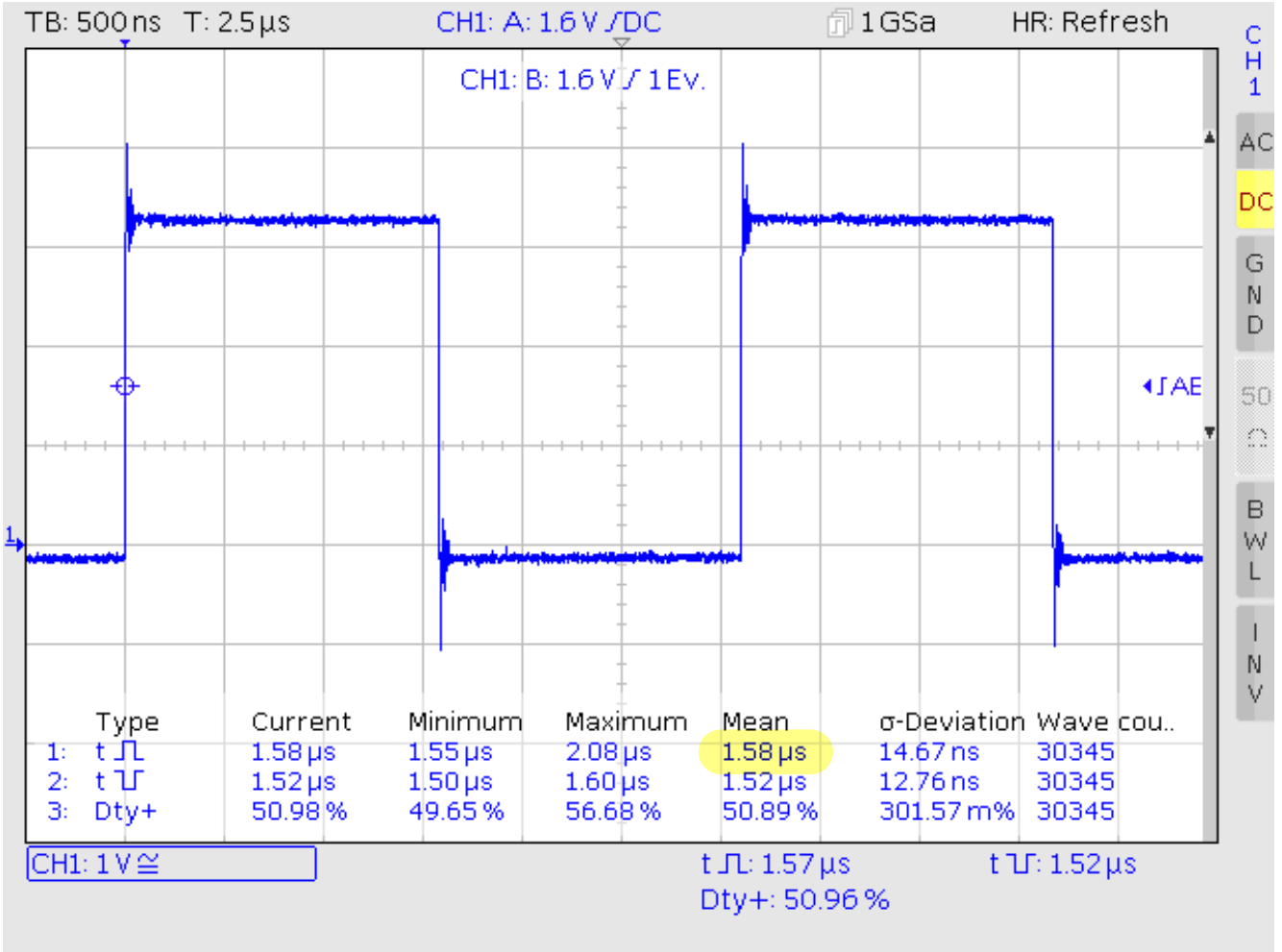
Toggeling of the GPIO in Userspace

```
void toggle_ioctl_v2_once(int fd, struct utoggle *utog)
{
        int ret, i;
        struct gpio_v2_line_values val;
        val.mask = 0x01;
        for (i = 0; i < 2; i++) {
                val.bits = i;
                ret = ioctl(fd, GPIO_V2_LINE_SET_VALUES_IOCTL, &val);
        }
}
void toggle_ioctl_v2(struct utoggle *utog)
{
        int ret, fd, i;
        struct gpio_v2_line_request req;
        fd = open(utog->devname, O_RDWR, 0);
        memset(&req, 0, sizeof(req));
        req.num_lines = 1;
        req.offsets[0] = utog->gpionr;
        strncpy(req.consumer, "toggle-ioctl-v2", sizeof(req.consumer)-1);
        req.config.flags = GPIO_V2_LINE_FLAG_OUTPUT;
        req.config.num_attrs = 0;
        ret = ioctl(fd, GPIO_V2_GET_LINE_IOCTL, &req);
        while (running) {
                for (i = 0; i < 5; i++)
                        toggle_ioctl_v2_once(req.fd, utog);
                usleep(1000);
        }
        close(fd);
}
```

Toggeling in userspace

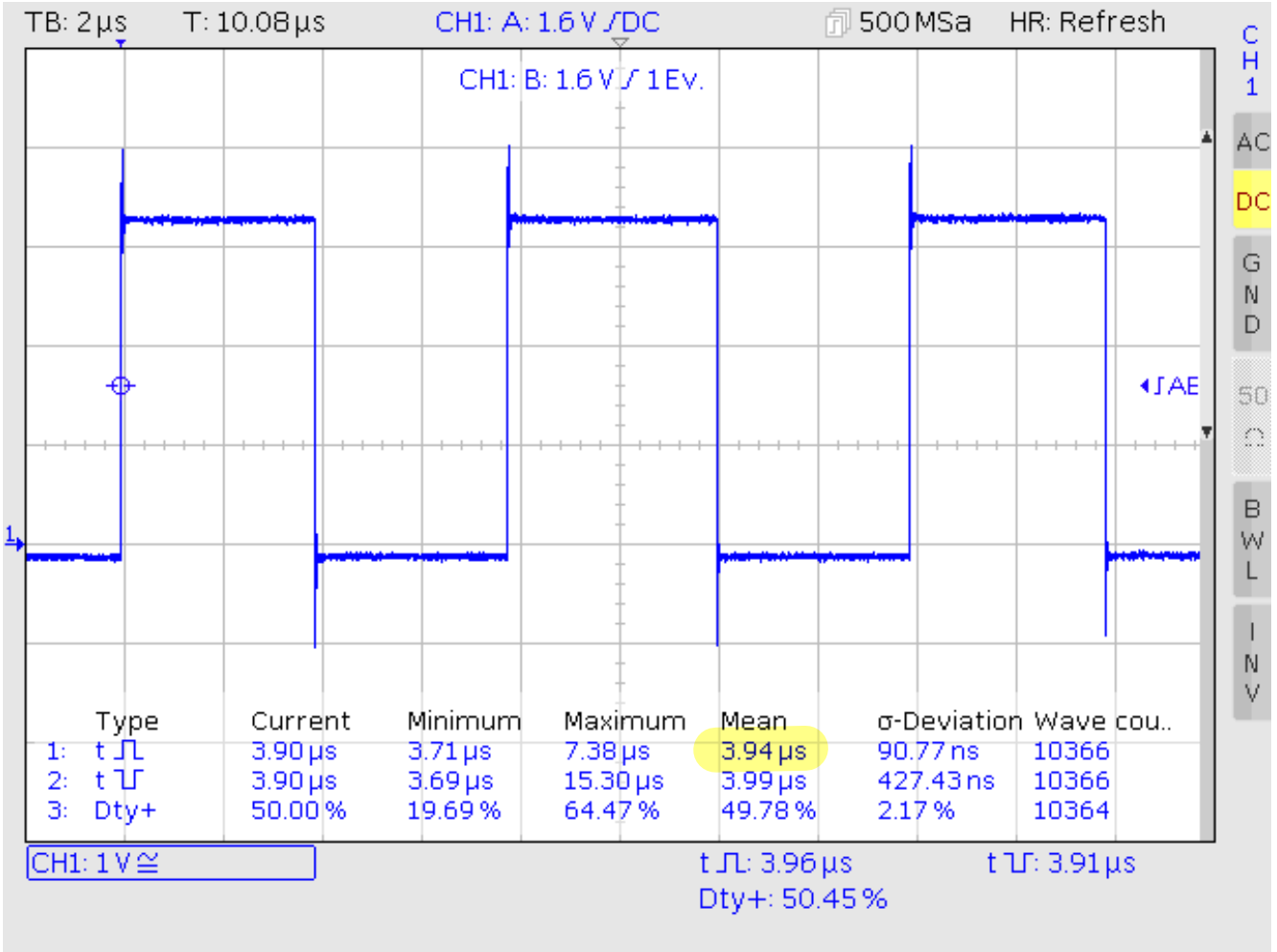
- Function `toggle_ioctl_v2()` runs as realtime userspace task
- `usleep()` is used for avoiding realtime throttling
- Measuring on the oscilloscope on the second call to `toggle_ioctl_v2_once()` of one phase after the `usleep()` $\rightarrow$ Measuring cache hot scenario
- Driver source file `utoggle.c` and all scripts used are available in the cloud
- Each measuring is done with a minimum of 10000 wave counts

Reference for userspace measurement




```
$ chrt -f 60 ./utoggle -I
```

Syscall events

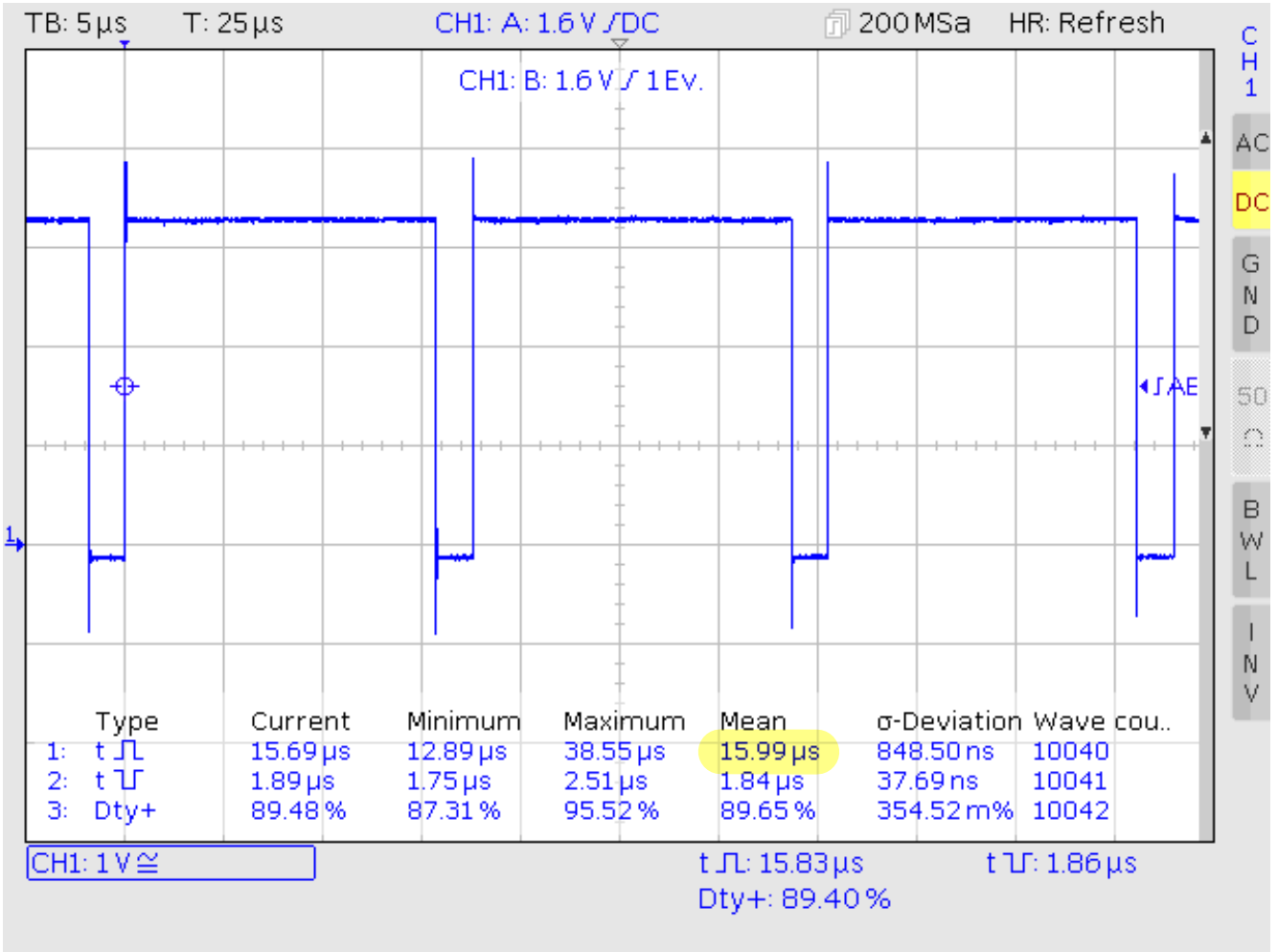


→ *Tracing event overhead:* 2.36μs (149%)

Syscall events

```
$ cd /sys/kernel/tracing
$ echo 1 > events/syscalls/sys_enter_ioctl/enable
$ echo 1 > events/syscalls/sys_exit_ioctl/enable

$ cat trace
utoggle-137 [000] 1540.593245:
            sys_ioctl(fd: 4, cmd: c010b40f, arg: bea08900)
utoggle-137 [000] 1540.593247: sys_ioctl -> 0x0
```

Uprobe event



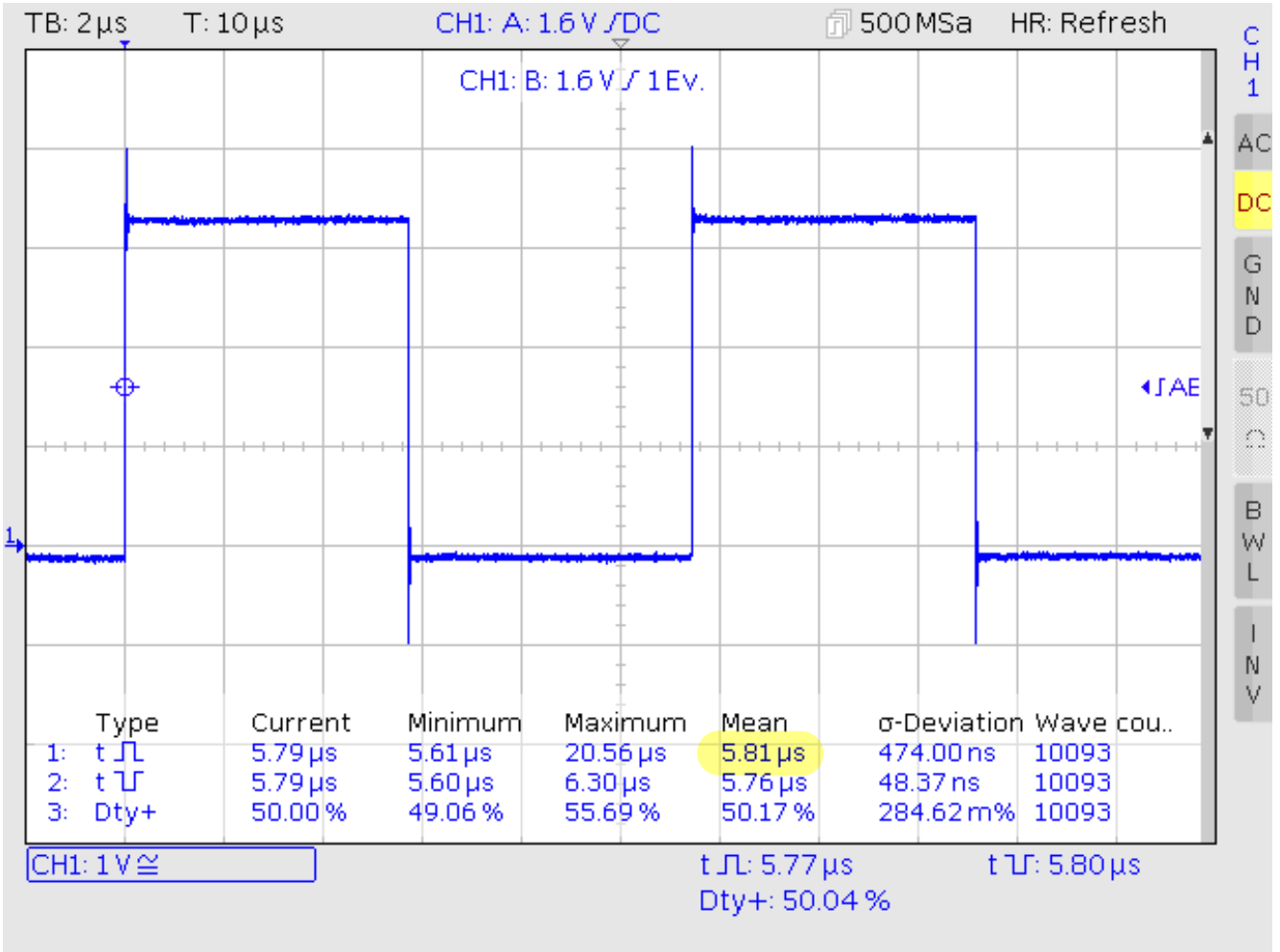
Monitoring only rising edge, not the falling one

→ *Tracing event overhead:* $14.41\,\mu s$ (912%)

Uprobe event

```
$ cd /sys/kernel/tracing

$ perf probe -x /root/utoggle
      -a 'toggle:ioctl=toggle_ioctl_v2_once fd=%r0:u32'

$ perf probe -x /root/utoggle
      -a 'toggle:ioctl=toggle_ioctl_v2_once%return ret=$retval:u32'

$ echo 1 > events/toggle/ioctl/enable
$ echo 1 > events/toggle/ioctl__return/enable

$ cat trace
utoggle-137 [000] 3620.485073: ioctl: (0x401174) fd=3
utoggle-137 [000] 3620.485117: ioctl__return:
                                 (0x401364 <- 0x401174) ret=0
```

Trace marker

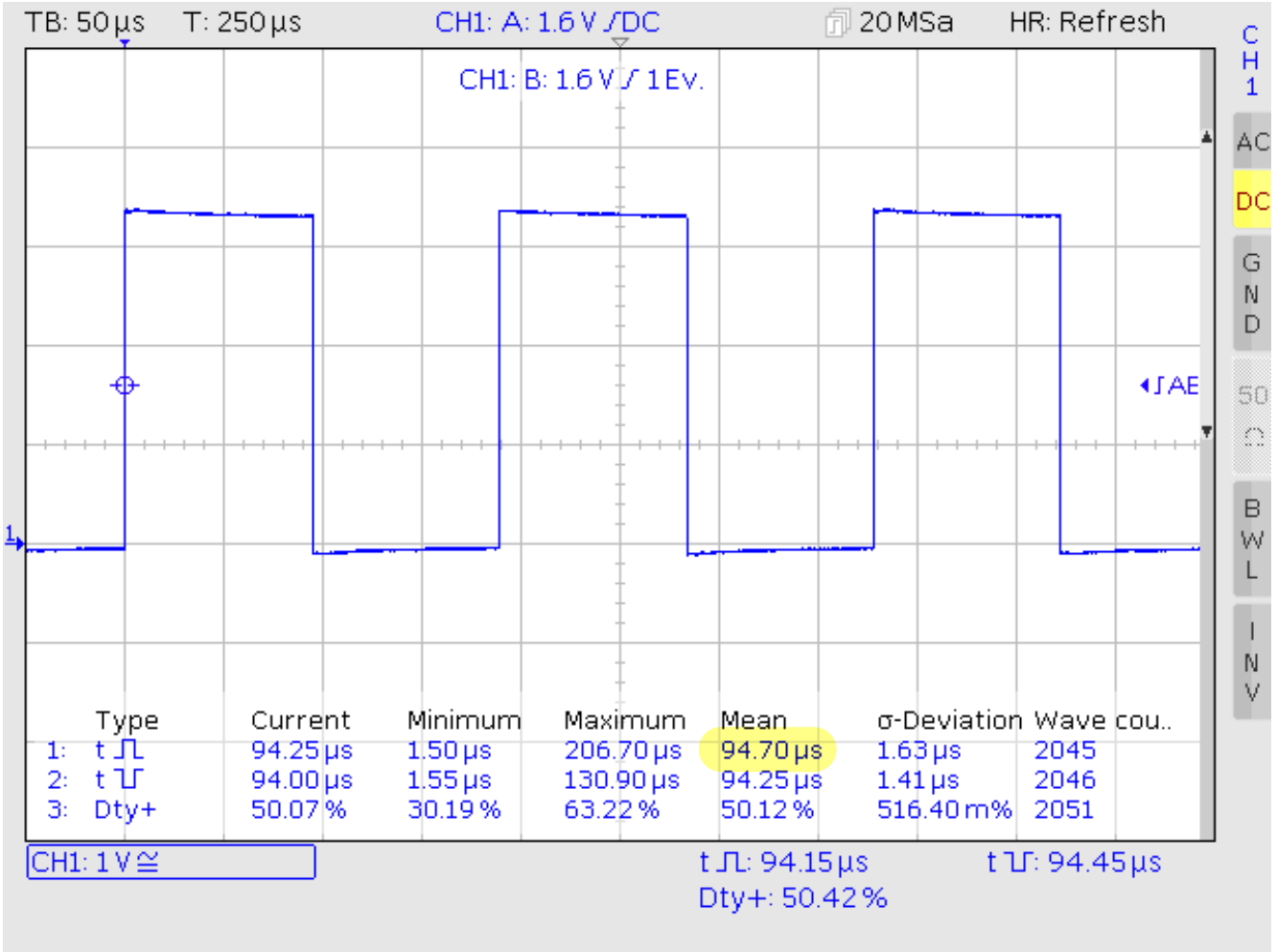


→ *Tracing event overhead:* 4.23μs (268%)

Trace marker

```
static int fd_tm = -1;

void trace_on(void)
{
        char *fmarker = "/sys/kernel/debug/tracing/tracing_on";
        if (fd_tm < 0)
                fd_tm = open(fmarker, O_WRONLY);
}

void trace_write(const char *fmt, ...)
{
        va_list ap;
        char buf[256];
        int n;
        va_start(ap, fmt);
        n = vsnprintf(buf, 256, fmt, ap);
        va_end(ap);
        write(fd_tm, buf, n);
}
```

Program strace

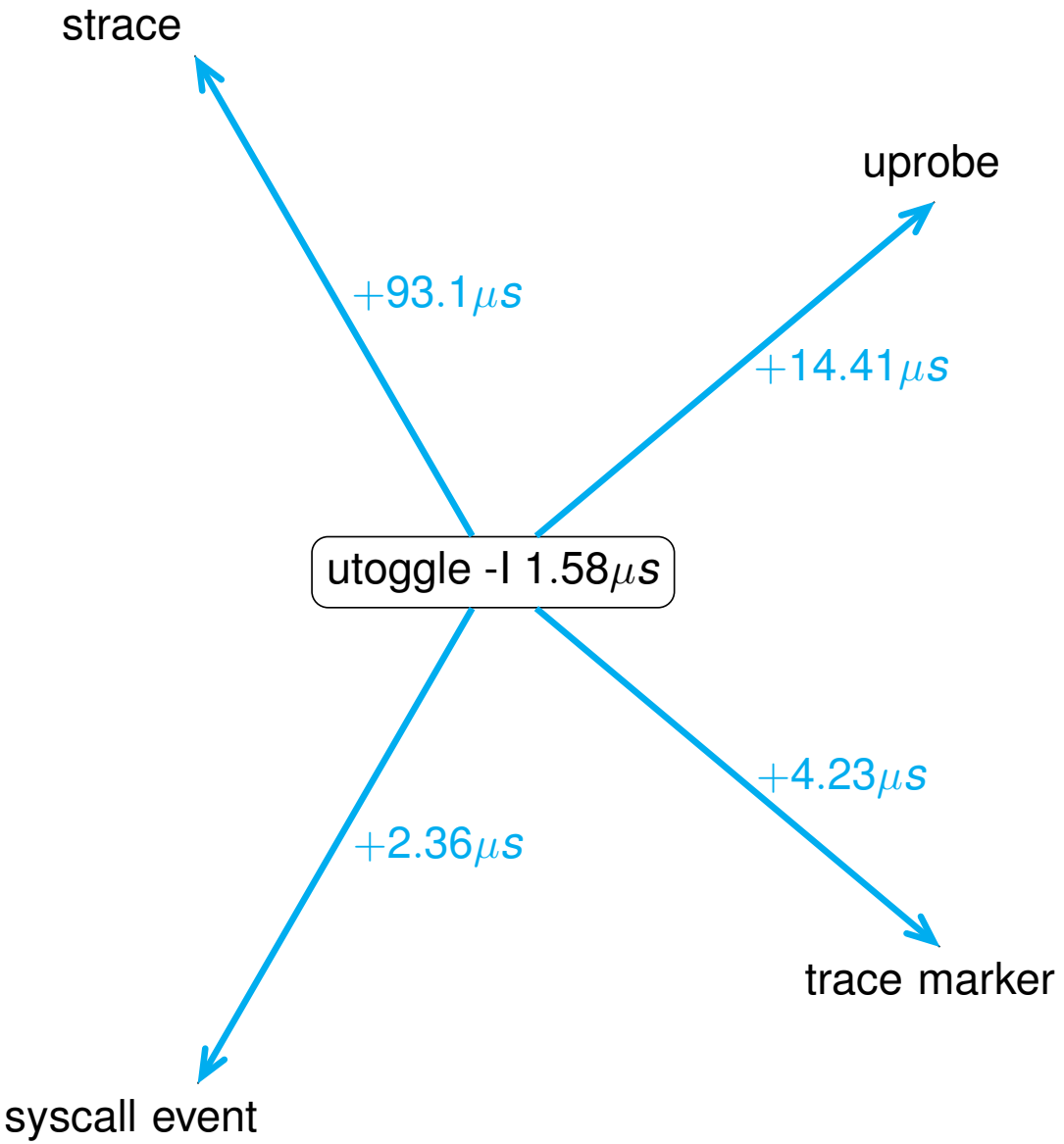


→ *Tracing event overhead:* 93.12μs (5894%)

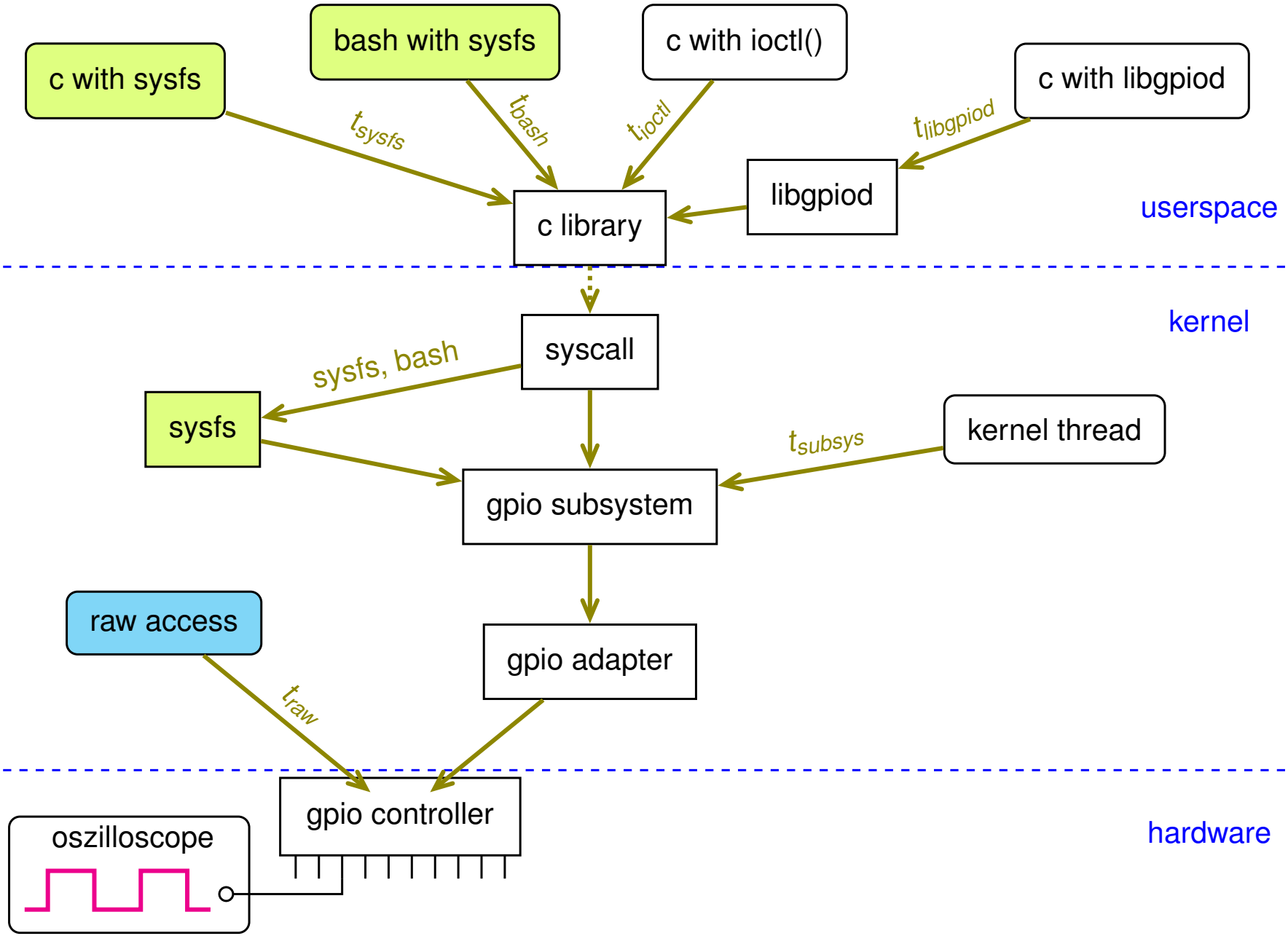
Program strace

```
$ pidof utoggle
188

$ strace -p 188
...
ioctl(4, GPIO_V2_LINE_SET_VALUES_IOCTL, {bits=0, mask=0x1}) = 0
ioctl(4, GPIO_V2_LINE_SET_VALUES_IOCTL, {bits=0x1, mask=0x1}) = 0
```

Userspace - measuring concept



Measuring gpio latencies



Toggle of gpio at max. frequency I

Variable	Description	Value
t_{raw}	Kernel thread in driver with raw register access	
t_{subsys}	Kernel thread in driver using regular gpio subsystem	
t_{ioctl}	`ioctl()` of `/dev/gpiochipN` in userspace	
$t_{libgpiod}$	C program uses libgpiod	
t_{sysfs}	C program uses sysfs	
t_{bash}	Shell script uses sysfs	

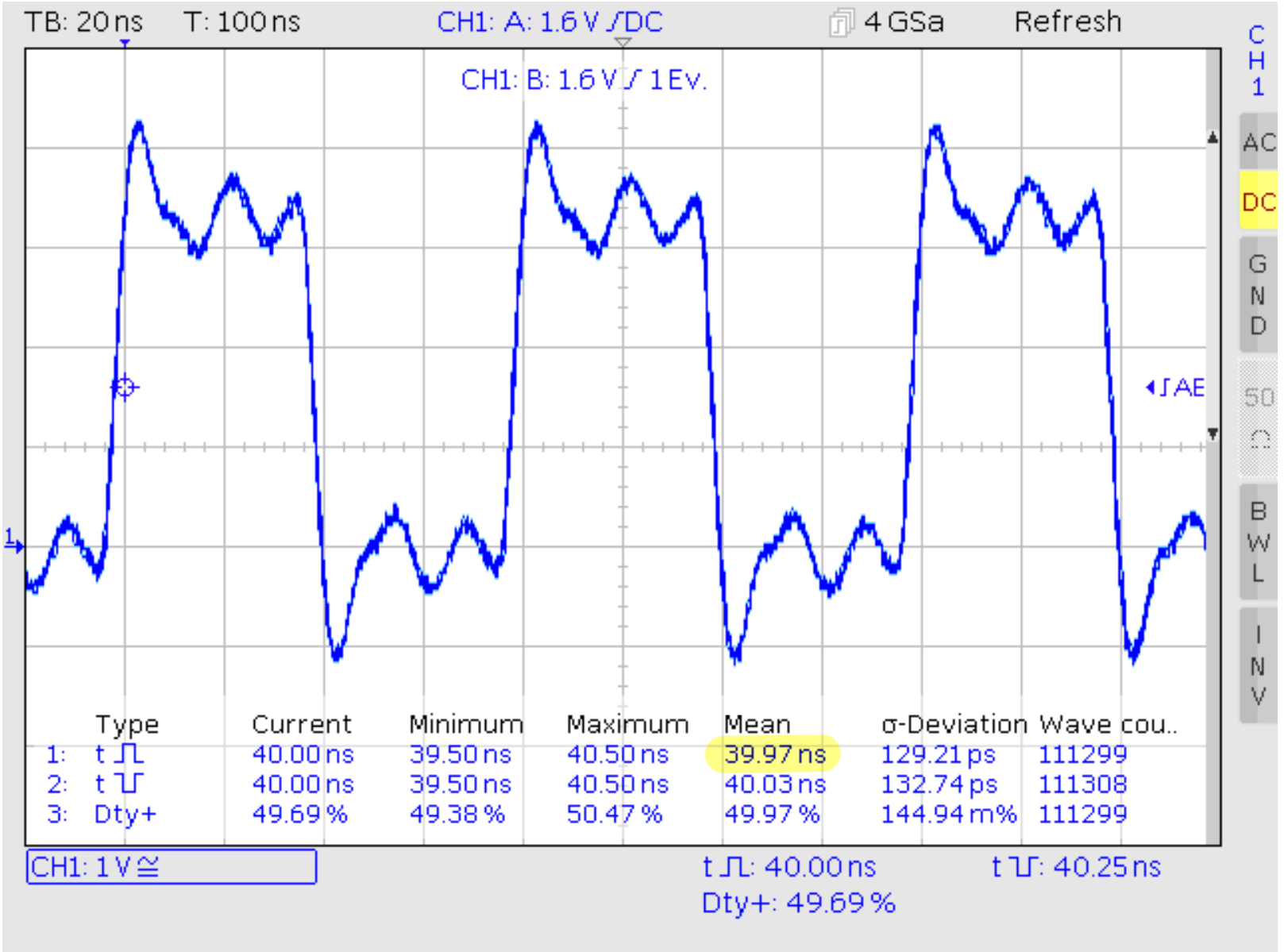
Toggle of gpio at max. frequency II

Calculation	Runtime (overhead)	Value
$t_{subsys} - t_{raw}$	Gpio subsystem	
$t_{ioctl} - t_{subsys}$	Syscall and vfs	
$t_{libgpiod} - t_{ioctl}$	Libgpiod	
$t_{sysfs} - t_{ioctl}$	Sysfs	
$t_{bash} - t_{sysfs}$	Bash	

Raw toggeling the gpio in driver

```
static int toggle_raw(void *arg)
{
    int i = 0;
    void __iomem *ctrl_base;
    void __iomem *gpio2_base;
    void __iomem *reg_value;
    u32 value, value_on, value_off;

    ctrl_base = ioremap( (unsigned int) 0x44E10000, 8192);
    gpio2_base = ioremap( (unsigned int) 0x481AC000, 4096);

    __raw_writel(0x0000000F, ctrl_base + 0x8A0);
    __raw_writel(__raw_readl(gpio2_base + 0x134) & 0xFFFFFFBF, gpio2_base + 0x134);
    value = __raw_readl(gpio2_base + 0x13C);
    value_on = value | 0x00000040;
    value_off = value & 0xFFFFFFBF;
    reg_value = gpio2_base + 0x13C;

    while (1) {
        if (i++ & 0x01)
            __raw_writel(value_on, reg_value);
        else
            __raw_writel(value_off, reg_value);
        if ((i & 0xFFFFF) == 0xFFFFF && kthread_should_stop())
            break;
    }
    iounmap(gpio2_base);
    iounmap(ctrl_base);
    return 0;
}
```

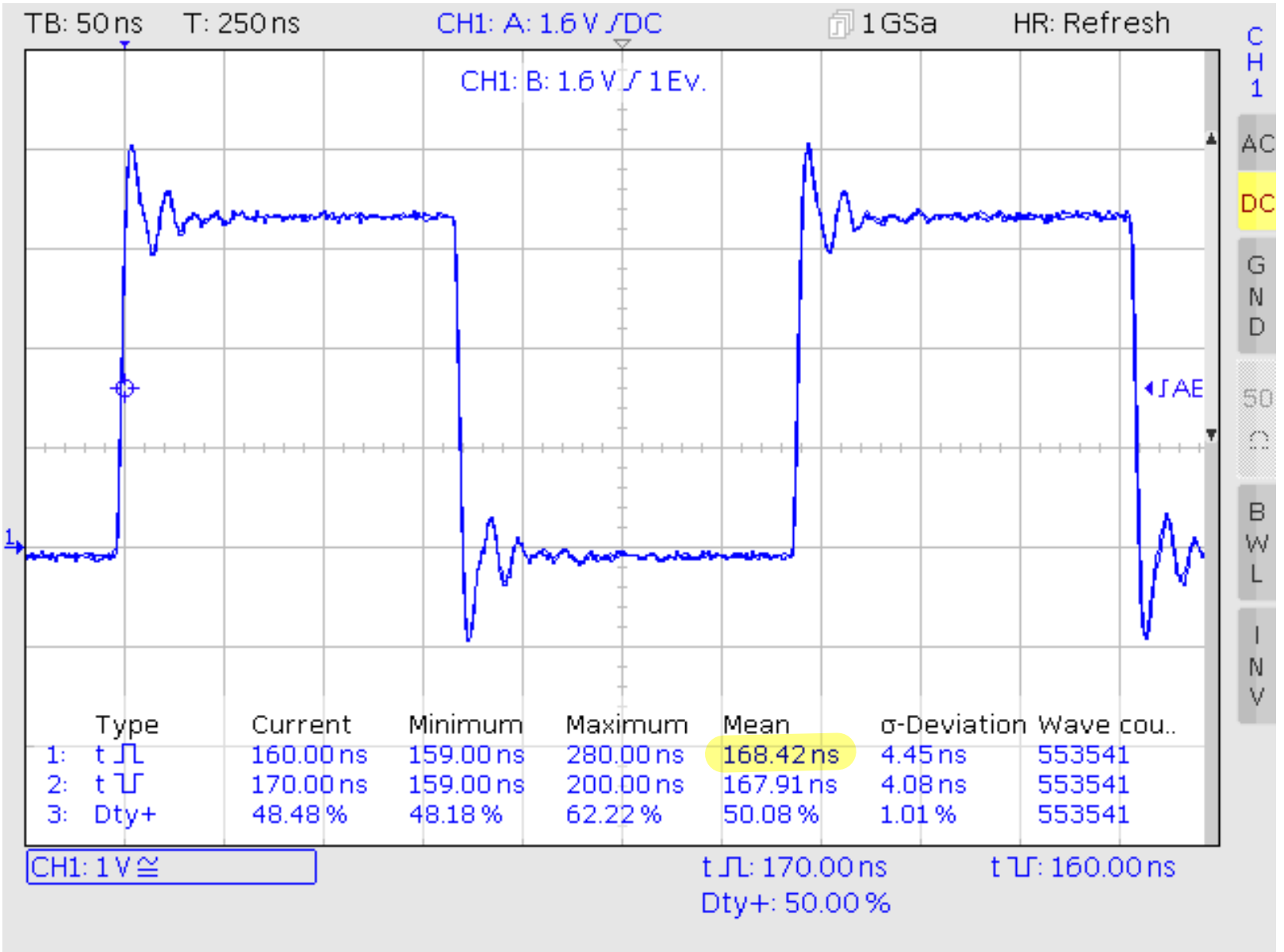
Toggeling of the gpio in driver with `raw_writel()`



Toggeling of the gpio in driver

```
struct toggle_data {
    struct device       *dev;
    struct gpio_desc    *gpiod_toggle;
};

static int toggle_gpio(void *arg)
{
    struct toggle_data *data = (struct toggle_data*)arg;
    int i = 0;

    while (1) {
        gpiod_set_value(data->gpiod_toggle, i++ & 0x01);
        if ((i & 0xFFFFF) == 0xFFFFF && kthread_should_stop())
            break;
    }

    return 0;
}
```

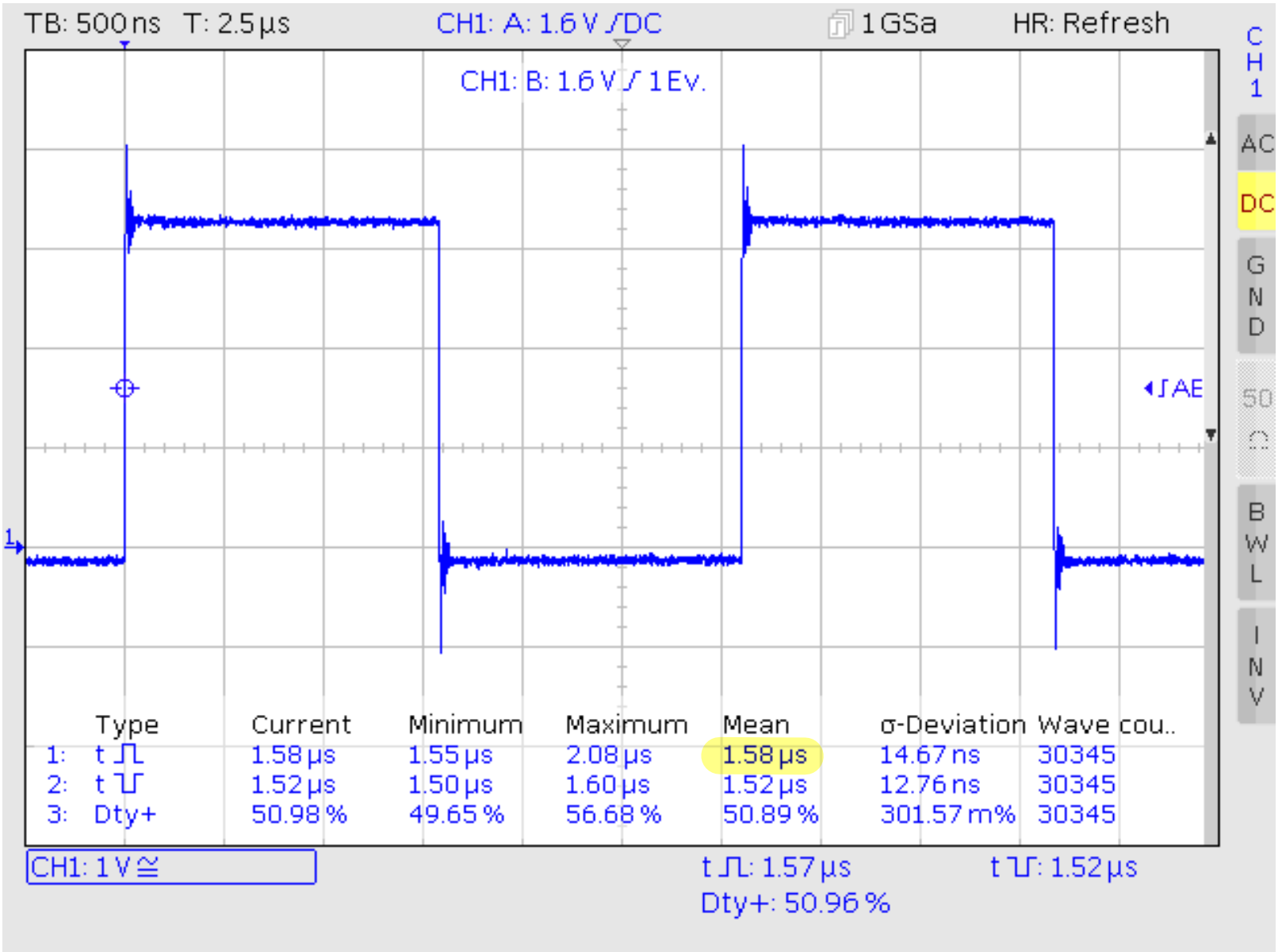
Toggeling of the gpio in driver



Toggle of gpio in userspace with `ioctl()`

```
int toggle_ioctl_v2(struct utoggle *utog)
{
    int ret, i, fd;
    struct gpio_v2_line_request req;
    struct gpio_v2_line_values val;
    int marker = utog->marker;
    fd = open(utog->devname, O_RDWR, 0);
    memset(&req, 0, sizeof(req));
    req.num_lines = 1;
    req.offsets[0] = utog->gpionr;
    strncpy(req.consumer, "toggle-ioctl-v2", sizeof(req.consumer)-1);
    req.config.flags = GPIO_V2_LINE_FLAG_OUTPUT;
    req.config.num_attrs = 0;
    ret = ioctl(fd, GPIO_V2_GET_LINE_IOCTL, &req);
    val.mask = 0x01;
    for (i = 0; running; i++) {
        val.bits = i; // & 0x01;
        ret = ioctl(req.fd, GPIO_V2_LINE_SET_VALUES_IOCTL, &val);
    }
    close(fd);
    return 0;
}
```

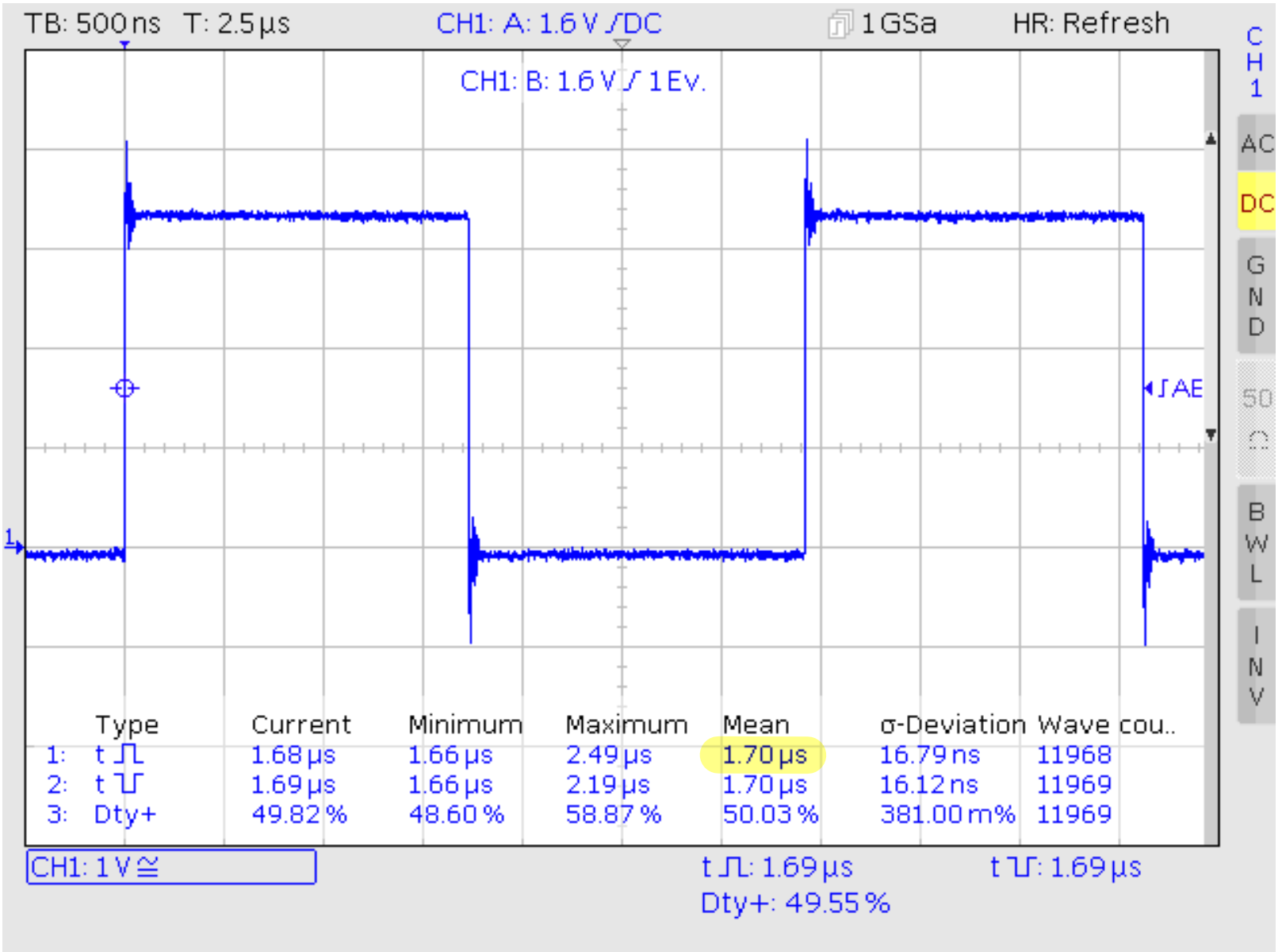
Toggle of gpio in userspace with `ioctl()` v2



Toggle of gpio in c program with libgpiod

```
int toggle_lib(struct utoggle *utog)
{
    int ret, i;
    struct gpiod_chip *chip;
    struct gpiod_line *line;
    int marker = utog->marker;
    chip = gpiod_chip_open(utog->devname);
    line = gpiod_chip_get_line(chip, utog->gpionr);
    gpiod_line_request_output(line, "toggle", 0);
    for (i = 0; running; i++) {
        ret = gpiod_line_set_value(line, i & 0x01);
    }
    gpiod_line_release(line);
    gpiod_chip_close(chip);
    return 0;
}
```

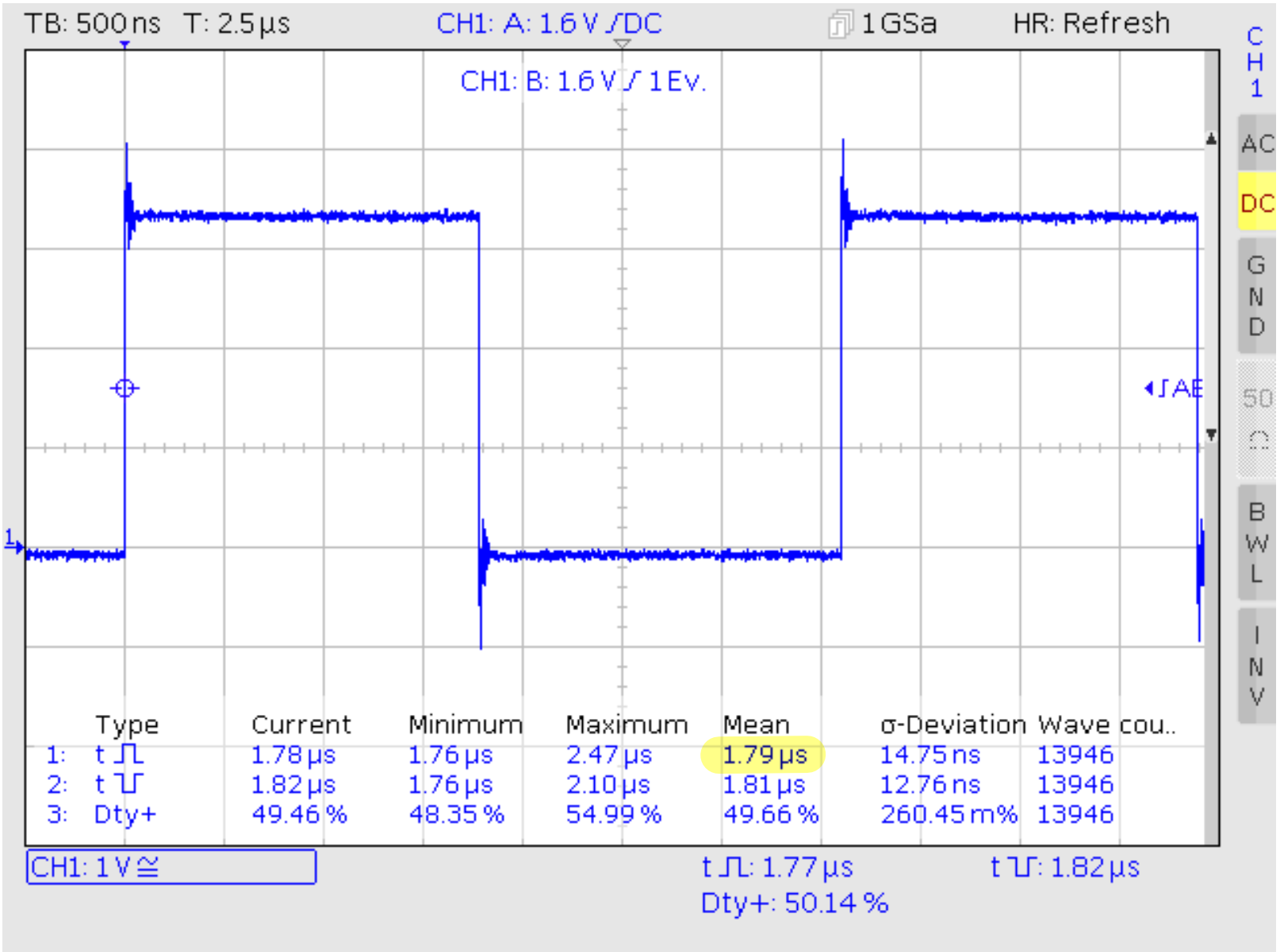
Toggle of gpio in c program with libgpiod



Toggle of gpio in sysfs with c program

```
int toggle_sysfs(struct utoggle *utog)
{
    int ret, i, fd;
    int marker = utog->marker;
    char fn[255];
    char buf[20];

    memset(buf, 0, sizeof(buf));
    sprintf(buf, "%d", utog->sysfsnr);
    fd = open("/sys/class/gpio/export", O_WRONLY);
    ret = write(fd, buf, strlen(buf));
    close(fd);
    memset(fn, 0, sizeof(fn));
    sprintf(fn, "/sys/class/gpio/gpio%d/direction", utog->sysfsnr);
    fd = open(fn, O_RDWR);
    ret = write(fd, "out", 3);
    close(fd);

    memset(fn, 0, sizeof(fn));
    sprintf(fn, "/sys/class/gpio/gpio%d/value", utog->sysfsnr);
    fd = open(fn, O_RDWR);
    for (i = 0; running; i++) {
        write(fd, "1", 1);
        write(fd, "0", 1);
    }
    close (fd);
    fd = open("/sys/class/gpio/unexport", O_WRONLY);
    ret = write(fd, buf, strlen(buf));
    close(fd);
    return 0;
}
```

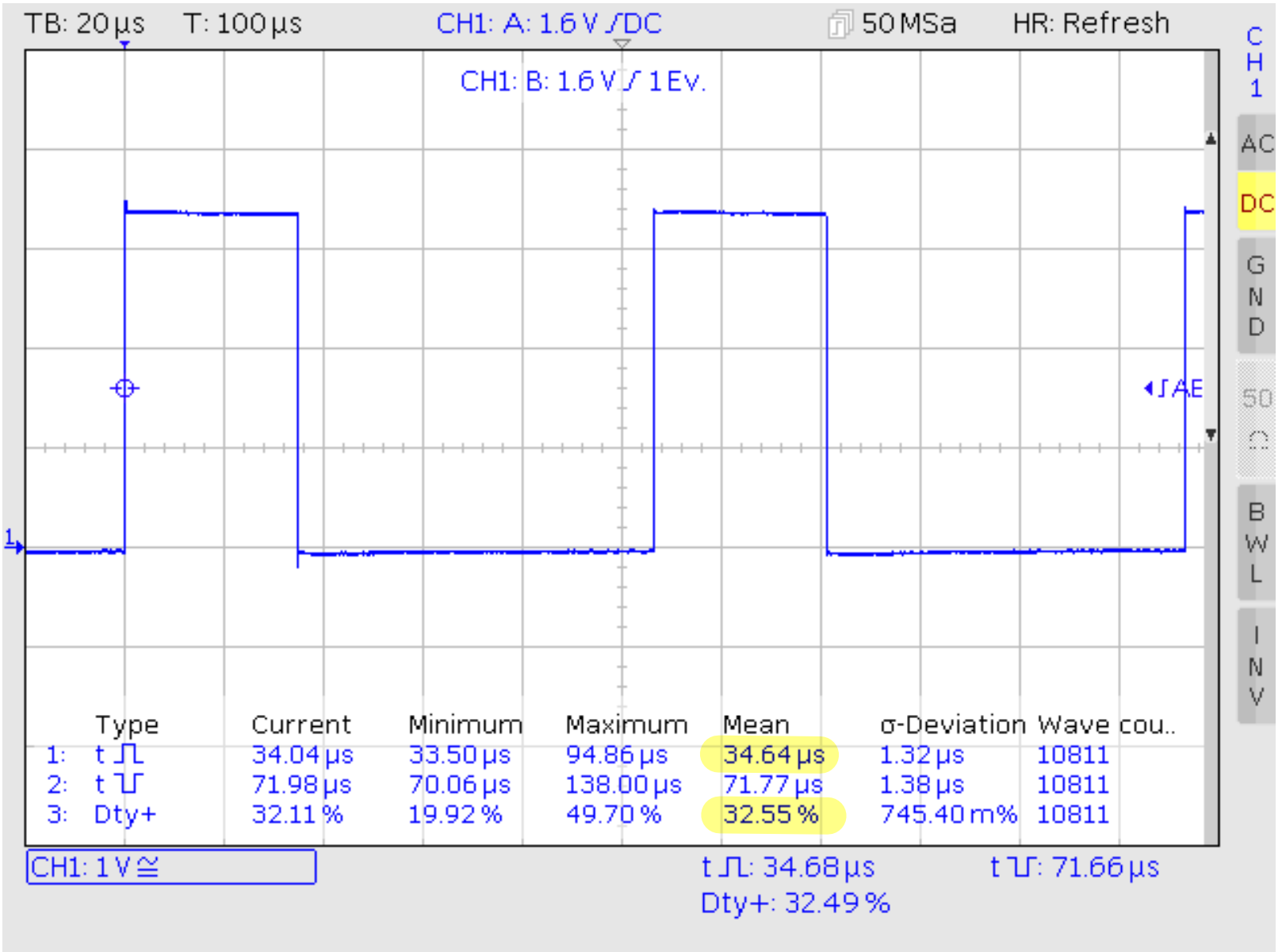
Toggle of gpio in sysfs with c program



Toggle of GPIO in sysfs with shell script

```
#!/bin/bash
dirg=/sys/class/gpio

[ -e $dirg/gpio70 ] || echo 70 > $dirg/export
echo out > $dirg/gpio70/direction

for ((i=0; i<1000000; i++))
do
    echo 1
    echo 0
done > $dirg/gpio70/value

echo 70 > $dirg/unexport
ls $dirg
```

Toggle of gpio in sysfs with shell script



Inhaltsverzeichnis I